НАЦІОНАЛЬНИЙ ТЕХНІЧНИЙ УНІВЕРСИТЕТ УКРАЇНИ
«КИЇВСЬКИЙ ПОЛІТЕХНІЧНИЙ ІНСТИТУТ
імені ІГОРЯ СІКОРСЬКОГО»
Факультет інформатики та обчислювальної техніки
(повне найменування інституту, факультету)
Автоматизованих систем обробки інформації і управління
(повна назва кафедри)

«До захисту допущено»
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис) (ініціали, прізвище)
“ ” 2020 р.

Дипломний проєкт

на здобуття ступеня бакалавра

за освітньо-професійною програмою «Програмне забезпечення інформаційних управляючих систем та технологій»

спеціальності «121 Інженерія програмного забезпечення»

на тему *Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля*

Виконав: студент IV курсу, групи *ІП-61 Денисенко Марк Олександрович* _____
(прізвище, ім’я, по батькові) (підпис)

Керівник *ст. вик. Недашківський Є. А.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Консультант з графічної документації *доц., к.т.н., Ліщук К.І.* _____
посада,науковий ступінь,вчене звання,прізвище,і ім’я, по батькові (підпис)

Рецензент:

доц. каф. ТК, к.т.н. Кисленко Ю. І.
посада,науковий ступінь,вчене звання,прізвище,ім’я, по батькові (підпис)

Засвідчую, що у цьому дипломному проєкті немає запозичень з праць інших авторів без відповідних посилань.
Студент _____
(підпис)

Київ – 2020 року

UNICHECK

Власник документу:
Попенко Володимир Дмитрович

Дата перевірки:
12.06.2020 15:46:11 EEST

Дата звіту:
12.06.2020 15:47:40 EEST

ID перевірки:
1004000089

Тип перевірки:
Doc vs Internet + Library

ID користувача:
77149

Назва документу: Denisenko_ip61

ID файлу: 1004013030 **Кількість сторінок:** 25 **Кількість слів:** 4595 **Кількість символів:** 34837 **Розмір файлу:** 37.68 KB

3.7% Схожість

Найбільша схожість: 1.39% з джерело бібліотеки. **ID файлу:** 1000017346

2.11% Схожість з Інтернет джерелами	80	Page 27
3.7% Текстові збіги по Бібліотеці акаунту	95	Page 27

0% Цитат

Не знайдено жодних цитат

0% Вилучень

Вилучений текст відсутній

Підміна символів

Не знайдено замінених символів

Національний технічний університет України
“Київський політехнічний інститут імені Ігоря Сікорського”

Факультет (інститут) *Інформатики та обчислювальної техніки*
(повна назва)

Кафедра *автоматизованих систем обробки інформації і управління*
(повна назва)

Рівень вищої освіти – перший (бакалаврський)

Спеціальність – *121 Інженерія програмного забезпечення*

Освітньо-професійна програма – *Програмне забезпечення інформаційних управляючих систем та технологій*

ЗАТВЕРДЖУЮ
В.о. завідувача кафедри
_____ *Олександр ПАВЛОВ*
(підпис)
“___”_____2020 р.

З А В Д А Н Н Я
НА ДИПЛОМНИЙ ПРОЄКТ СТУДЕНТУ

Денисенку Марку Олександровичу
(прізвище, ім’я, по батькові)

1. Тема проєкту «*Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля*»

керівник проєкту *Недашківський Євген Анатолійович, старший викладач*
(прізвище, ім’я, по батькові, науковий ступінь, вчене звання)
затверджені наказом по університету від “07” травня 2020 р. №1081-с

2. Термін подання студентом проєкту «08» *червня 2020 року*

3. Вихідні дані до проєкту
Технічне завдання

4. Зміст пояснювальної записки
1) Аналіз вимог до програмного забезпечення: основні визначення та терміни, опис предметного середовища, огляд існуючих технічних рішень та відомих програмних продуктів, розробка функціональних та нефункціональних вимог
2) Моделювання та конструювання програмного забезпечення: моделювання та аналіз програмного забезпечення, засоби розробки, технічні рішення, архітектура програмного забезпечення
3) Розгортання та впровадження програмного забезпечення
4) Керівництво користувача, методика випробувань програмного продукту

5. Перелік графічного матеріалу

1) Схема структурна варіантів використань

2) Схема бази даних

3) Креслення вигляду екранних форм

6. Консультанти розділів проєкту

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата	
		завдання видав	завдання прийняв

7. Дата видачі завдання «10» *березня 2020 року*

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів виконання дипломного проєкту	Термін виконання етапів проєкту	Примітка
1.	*Вивчення рекомендованої літератури*	*06.04.2020*	
2.	*Аналіз існуючих методів розв'язання задачі*	*10.04.2020*	
3.	*Постановка та формалізація задачі*	*13.04.2020*	
4.	*Аналіз вимог до програмного забезпечення*	*16.04.2020*	
5.	*Алгоритмізація задачі*	*22.04.2020*	
6.	*Моделювання програмного забезпечення*	*29.04.2020*	
7.	*Обґрунтування використовуваних технічних засобів*	*05.05.2020*	
8.	*Розробка архітектури програмного забезпечення*	*08.05.2020*	
9.	*Розробка програмного забезпечення*	*14.05.2020*	
10.	*Налагодження програми*	*18.05.2020*	
11.	*Виконання графічних документів*	*21.05.2020*	
12.	*Оформлення пояснювальної записки*	*25.05.2020*	
13.	*Подання ДП на попередній захист*	*28.05.2020*	
14.	*Подання ДП рецензенту*	*01.06.2020*	
15.	*Подання ДП на основний захист*	*08.06.2020*	

Студент _____ Марк Денисенко
(підпис)

Керівник _____ Євген Недашківський
(підпис)

№з/п	Формат	Позначення	Найменування	Кількість листів	Примітка
1	А4		Завдання на дипломний проєкт	2	
2	А4	КПІ.ІП-6108.045490.02.81	Пояснювальна записка	61	
3	А4	КПІ.ІП-6108.045490.03.91	Технічне завдання	12	
4	А4	КПІ.ІП-6108.045490.04.51	Програма та методика тестування	6	
5	А4	КПІ.ІП-6108.045490.05.13	Опис програми	29	
6	А4	КПІ.ІП-6108.045490.06.34	Керівництво користувача	7	
7	А3	КПІ.ІП-6108.045490.07.СБД	Схема бази даних	1	
8	А3	КПІ.ІП-6108.045490.07.СС	Схема структурна варіантів використань	1	
9	А3	КПІ.ІП-6108.045490.07.КЕ	Креслення вигляду екранних форм	1	

Зм.	Арк.	ПІБ	Підп.	Дата	КПІ.ІП-6108.045490.00.90		
Розробн.		Денисенко М. О.			Відомість дипломного проєкту	Літ.	Лист 1 / Листів 1
Керівн.		Недашківський Є. А.					
Консульт.						КПІ ім.Ігоря Сікорського кафедра АСОІУ гр. ІП-61	
Н/контр.		Ліщук К.І.					
В.о.зав.каф.		Павлов О.А.					

АНОТАЦІЯ

Структура та обсяг роботи. Пояснювальна записка дипломного проєкту складається з чотирьох розділів, містить 120 сторінок, 23 рисунки, 22 таблиці, 5 додатків, 11 джерел.

Об'єкт дослідження: Дипломний проект присвячений побудові геоінформаційної системи надання рекомендацій з пошуку місць дозвілля.

Мета дипломного проекту: створити геоінформаційну систему надання рекомендацій з пошуку місць дозвілля.

У першому розділі, до якого входять загальні положення, опис предметної області та порівняння з існуючими рішеннями, було описано проблему для вирішення якої створено програмне забезпечення, також проаналізовано аналоги та їх недоліки з метою поліпшення кінцевого продукту та наведено основні вимоги до функціональності програмного забезпечення.

Другий розділ пояснює технічну сторону розробки та використані підходи при написанні кодової бази. Також включає в себе схеми бізнес-процесів взаємодії з сервісом.

Розділ якості програмного забезпечення описує способи та тип тестування. До цієї частини входить покрокова демонстрація контрольного прикладу.

КЛЮЧОВІ СЛОВА: ІНФОРМАЦІЙНА СИСТЕМА, ГЕОЛОКАЦІЯ, ДОЗВІЛЛЯ, ПОШУК.

ABSTRACT

Structure and scope of work. The explanatory note of the graduation project consists of four sections, containing 120 pages, 23 pictures, 22 tables, 5 applications, 11 sources.

Object of research: The diploma project is devoted to the construction of a geographic information system for providing recommendations for finding leisure places.

The purpose of the diploma project: to create a geographic information system to provide recommendations for finding leisure places.

The first section, which includes general provisions, a description of the subject area and a comparison with existing solutions, described the problem for which the software was created, also analyzed analogues and their shortcomings to improve the final product and presented the basic requirements for software functionality.

The second section explains the technical side of the development and the approaches used in writing the code base. Also includes schemes of business processes of interaction with service.

The software quality section describes the methods and type of testing. This part includes a step-by-step demonstration of a control example.

KEYWORDS: INFORMATION SYSTEM, GEOLOCATION, LEISURE, SEARCH.

.

Пояснювальна записка
до дипломного проєкту

на тему: ГЕОІНФОРМАЦІЙНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ НАДАННЯ РЕКОМЕНДАЦІЙ З ПОШУКУ МІСЦЬ ДОЗВІЛЛЯ

Київ – 2020 року

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ, СИМВОЛІВ, ОДИНИЦЬ, СКОРОЧЕНЬ І ТЕРМІНІВ

ІС – це інформаційна система.

ПЗ – це програмне забезпечення.

EF – об’єктно реляційне програмне забезпечення для взаємодії з базою даних.

SPA – веб-сайт, що працює без повного перезавантаження веб-сторінки, наближає відчуття користувача до роботи з настільною програмою.

API (Прикладний програмний інтерфейс) – це інструмент взаємодії між програмою та апаратним забезпеченням, що надає можливість використання споживчого застосунку.

Тег – є змістовною ознакою даних, що помічає певну сукупність інформації, дозволяючи у такий спосіб секціонувати дані за певними критеріями та характерними деталями. Створені з метою полегшення пошуку інформації.

Юніт-тест – це найменший з різновидів програмних тестів, що відповідає за перевірку правильності логіки найменших частин коду ізольовано від впливу інших частин програмного забезпечення.

ВСТУП

Існування людини є складним процесом, що включає багато різних аспектів життя. Основними з них є робота або ж навчання, фізичний та моральний відпочинки. Розглянувши та проаналізувавши останній аспект, можна зрозуміти, що найбільше зусиль покладається на те, щоб в результаті мати більше вільного часу та змогу витрачати його у бажаний спосіб.

Для вдоволення людських потреб постійно виникають нові або ж змінюються вже існуючі місця дозвілля. Раніше невідомі заклади викликають неабиякий інтерес. Люди прагнуть дізнатися більше про те, чого вони до цього моменту не знали. У цьому випадку на допомогу приходять веб-сервіси з можливістю перегляду відгуків та рекомендацій про певні місця та заклади.

В наш час уже існують спеціалізовані ресурси та платформи для планування довготривалих мандрівок та пошуку відомих місць відпочинку. Але також залишається великий прошарок короткотривалого дозвілля та тимчасових подій про які не завжди вдається вчасно дізнатися через затяжний та складний процес додавання нових можливостей для відпочинку на відповідні сервіси.

Тому взявши до уваги описану проблему складності поширення маловідомих та не дуже популярних подій, виникає потреба у створенні інформаційної системи для надання користувачам можливості самостійно поширювати та дізнаватися інформацію з рекомендаціями про різні місця дозвілля беручи до уваги географічне розташування людей, для полегшення пошуку бажаних за змістом закладів поруч з необхідним географічним положенням.

Метою обраного дипломного проєкту є розробка інформаційної системи для полегшення пошуку інформації про місця дозвілля у розрізі геолокації.

1 АНАЛІЗ ВИМОГ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

1.1 Загальні положення

Цифрова трансформація світу, насамперед, спрямована на спрощення пошуку інформації та її прикладного використання у житті. Оскільки відпочинок є однією з найбільш важливих частин людського існування, то сервіси, що допомагають вирішити проблему пошуку дозвілля, користуються великим попитом.

Людство постійно стає більш вибагливішим до нових видів проведення вільного часу то ж для вдоволення цих потреб виникають нові місця та заклади. Поширення інформації про розташування давно перейшло від фізичного підходу, використовуючи рекомендації знайомих, рекламні банери та листівки, до нового цифрового формату, що під поширенням розуміє додавання інформації на спеціалізовані ресурси або ж створювати власні сервіс в мережі Інтернет.

Зазвичай найбільш правдивим джерелом є інші люди, які вже відвідали місце і мають повне та чітке уявлення, яким можуть поділитися, не додаючи неправдивих фактів та перебільшень з маркетинговою ціллю. Тому пріоритетною задачею є надання користувачам можливості швидко створювати нові місця та записи про них, а також кожен мусить мати змогу переглядати вже існуючі записи у розрізі географічного розташування.

Місця відпочинку поділяються за типом, тому для користувачів також важливо мати змогу робити пошук або ж отримувати рекомендації за бажаним змістом або комбінацією суттєвих ознак.

Тож перед початком розробки, було проаналізовано базові потреби аудиторії та продумано методи полегшення пошуку місць дозвілля. Усі важливі фактори, включно користувачів потреб до безпеки, були прийняті до уваги під час розгляду основних положень.

1.2 Змістовний опис і аналіз предметної області

Місцем дозвілля можна вважати будь-яке місце, де людина захоче провести вільний час. То ж це може бути як і невеликий сквер або спортмайданчик, так і головний стадіон країни чи дорогий ресторан.

Оскільки це достатньо широке поняття, то існує потреба розділяти такі місця за змістом або суттєвими ознаками. Зазвичай використовується самостійна класифікація від сервісу за відношенням до певних категорій, але така модель позбавляє гнучкості. Тому надавши можливість користувачам самостійно позначати місця змістовними маркуваннями допоможе удосконалити механізми пошуку необхідних закладів.

Для позначення географічної позиції прийнято використовувати широту та довготу, що представляють кути між екваторіальною площиною та нульовим меридіаном. Комбінація цих двох характеристик дозволяє з високою точністю ідентифікувати географічне положення об’єкту, не враховуючи висоту та глибину. Оскільки останні параметри у більшості випадків є надлишковими, то їх не враховують, а для конкретної ідентифікація використовують можливість вказати адресу або ж назву необхідного закладу.

Такий підхід створює можливість описувати та залишати відгук про будь-які місця, навіть якщо вони розташовані в одній будівлі, але на різних поверхах. Сам допис включає в себе текстове повідомлення, мітку вдоволення або ж навпаки, набір змістовних тегів та дату створення, щоб залишити можливість орієнтуватися в актуальності повідомлення серед інших.

Звісно, що кожен запис буде суб’єктивним відображення реальності, тому для виділення найбільш об’єктивних мусить бути наявна система підтримки інформативних та правдивих відгуків.

Оскільки світ достатньо великий, то для зручності користувача має бути наявна можливість відслідковувати початкове географічне розташування з метою показу релевантних записів до положення на мапі.

1.3 Аналіз успішних ІТ-проєктів

1.3.1 Аналіз відомих технічних рішень

При побудові систем пов’язаних зі зберіганням просторових даних виникають інші потреби до можливостей їх фільтрації та вибірки. Для роботи з даними географічного розташування найрозповсюдженими структурами оптимізації є R-tree, Geohash, Дерево квадрантів та Дерево октантів. Ці та деякі більш спеціалізовані типи підтримуються окремими так званими просторовими базами даних, а також деякими NoSQL і навіть SQL.

Але для початку обрано Microsoft SQL Server, який має вбудований тип даних для таких випадків і використовує B-tree. Цього більш ніж достатньо для невеликого застосунку, але у подальшому робота з координатами може бути винесена у більш спеціалізовану БД як наприклад Geocouch, CartoDB, SpatialDB або SpaceBase.

Для відображення мапи та створення можливості інтерактивної взаємодії користувача з нею краще за все використати готові рішення з продуманими механізмами. Найпопулярнішими є:

- OpenStreetMap;
- Mapbox;
- Google Maps API.

OpenStreetMap - відкрите програмне забезпечення для вільних даних про географічну місцевість та просторові дані. Не має на меті комерційну ціль та повністю підтримується волонтерами та суспільством сформованим навколо проєкту. Як наслідок не має гнучких та надійних методів інтеграції зі сторонніми сервісами, зміни відображення та стилю, додавання нового функціоналу. Перевагою використання є незалежність від будь-якої компанії та довготривала можливість з обслуговування. Також велика спільнота зможе дати відповідь на більшість питань, що можуть виникнути в процесі розробки.

Mapbox - комерційне програмне забезпечення для візуалізації географічної мапи та об’єктів на ній з можливостями кастомізації та полегшеними способом інтеграції зі сторонніми системами. Має безкоштовний план використання для мало споживчих та невеликих застосунків. Все більше набирає популярності, бо має найнижчу цінову політику серед конкурентів та перевірену роками якість і репутацію. Серед компаній, котрі вже користуються послугами сервісу є Uber, Snapchat, Facebook, Yahoo та багато інших. Це слугує ознакою надійності та довіри особливо між гігантами IT індустрії.

Google Maps API - найпоширеніший сервіс для графічного відображення мапи та усіх супутніх варіантів використання просторових даних. Має найширші можливості інтеграції та глобалізації для найрізноманітніших культур та національностей. Досить гнучке стилістичне супроводження з можливістю зміни найдрібніших деталей на власні. До простої мапи досить просто може бути інтегровано ще ряд супутніх сервісів з метою розширення можливостей системи, а саме: Places API, що надає можливість отримувати місця поруч з географічною координатою та інформацію про них, Geocoding API, сервіс для трансформування просторових даних до зрозумілих людині назв вулиць та місцевостей. Має можливість випробування усіх існуючих сервісів на випробувальний період шляхом надання бонусних коштів новим користувачам.

Тому взявши до уваги усі наведені факти відносно можливих технічних засобів реалізації інформаційної системи, було обрано Google Maps API оскільки має найбільший спектр доступних рішень для різних типів діяльності та найкращу підтримку своїх сервісів. На перших етапах розробки важливим є швидкість створення забезпечення та її гнучкість, що є запропонованим в останньому варіанті.

1.3.2 Аналіз відомих програмних продуктів

Проаналізувавши наявні програмні продукти, що існують на ринку та вирішують схожі проблеми, біли виявлені такі рішення:

- 2GIS;
- Yandex Maps;
- Tripadvisor.

2GIS – міжнародний картографічний сервіс, що надає послуги довідника. Вміщує достатньо великий обсяг інформації про об’єкти, заклади та місця дозвілля. Надає користувачам початкову контактну інформацію, опис та можливість залишати рекомендацію або ж відгук. Оскільки вся інфраструктура включно з деталями мапи контролюється утримуючою компанією, то можливість додати власні місця та записи про них відсутня. Також це найбільш прибуткова система серед існуючих картографічних рішень через найкращу монетизацію, тому рекомендації у цій системі не завжди є цілком об’єктивними та можуть носити рекламний характер, що може стати причиною вибору дозвілля інакшою якості від очікуваного.

Yandex Maps – російський картографічний сервіс, що входить до екосистеми Yandex. Володіє широким спектром можливостей пошуку місць дозвілля та закладів відпочинку, а саме за назвою, категорією, географічним розташуванням за координатою. Має деталізовану мапу усього світу. Також наявна система відгуків та рекомендацій. Можна підтримувати повідомлення інших користувачів вподобаннями.

Tripadvisor – широковідома платформа для пошуку та бронювання готелів, ресторанів, історичних пам’яток та деяких інших місць дозвілля. В ній наявна система відгуків та оцінки закладів, підтримки коментарів інших спеціалістів тощо. Але для додавання нового місця до інформаційної системи потрібно пройти процес додавання зі сторони розробників та утримувачів проєкту. Також поділ на категорії відбувається за власними критеріями і їх комбінування неможливе. Це викликає складнощі при використані платформи з метою пошуку короткочасних розваг або нових закладів.

Тож підсумовуючи проблеми існуючих рішень можна допрацювати проблемні аспекти, та додавши зручний інтерфейс для гнучкого пошуку,

утвориться корисний продукт для повсякденного використання з метою вирішення проблеми відпочинку.

1.4 Аналіз вимог до програмного забезпечення

Інформаційна система включає у себе два типи користувачів: авторизований та гість. На концептуальному рівні, перший тип є основним оскільки створює записи про різні місця дозвілля, бере участь в оцінці дописів інших користувачів та може відстежувати статистику власних. В той час як гість, є користувачем, що заходить до системи з метою пошуку необхідної інформації для власних потреб у проведенні дозвілля. Інтерфейс головною сторінки є схожим для обох типів, але гість має урізаний функціонал взаємодії з системою.

1.4.1 Розроблення функціональних вимог

До основних вимог інформаційної системи відносяться:

- реєстрація користувача та його авторизація у системі;
- створення записів з географічною прив’язкою у просторі;
- перегляд існуючих записів у розрізі певного географічного положення;
- для зареєстрованих користувачів перегляд статистики активності по власним записам.

Веб-сервіс може бути використаний за нижченаведеними варіантами використання у таблицях 1.1 – 1.10.

Таблиця 1.1 - Варіант використання "Реєстрація"

Назва	Реєстрація
Опис	Кожен користувач має змогу зареєструватися в системі
Актор	Анонім
Передумови	-
Постумови	Новий профіль в системі

Продовження таблиці 1.1

Сценарій	Користувач переходить на сторінку реєстрації Заповнює поля: Ім’я, Логін, Пошта, Пароль, Номер телефону, Стать, Країна Натискає кнопку зареєструватися Очікує верифікації сервером та перенаправлення на головну сторінку
Розширені сценарії	- Користувач увів не всі потрібні дані, то система підсвічує потрібні для заповнення поля - Користувач увів вже існуючі дані для деяких унікальних полів і система повідомляє про необхідність їх зміни

Таблиця 1.2 - Варіант використання "Автентифікація"

Назва	Автентифікація
Опис	Кожен зареєстрований користувач має змогу автентифікуватися в системі
Актор	Анонім
Передумови	Наявний профіль в системі
Постумови	Автентифікований користувач
Сценарій	Користувач переходить на сторінку входу до системи Заповнює поля: Логін та Пароль від існуючого профілю Натискає кнопку «Увійти» Очікує верифікації сервером та перенаправлення на головну сторінку
Розширені сценарії	- Користувач увів не всі потрібні дані, то система підсвічує потрібні для заповнення поля - Користувач неправильні дані для деяких полів і система повідомляє про необхідність їх зміни

Таблиця 1.3 - Варіант використання "Створення запису"

Назва	Створення запису
Опис	Кожен зареєстрований користувач має змогу створити запис
Актор	Користувач
Передумови	Наявний профіль в системі, автентифікований користувач
Постумови	Новий запис додано до системи
Сценарій	Користувач переходить на сторінку з мапою Заповнює обов'язкове поле «Текст» та необов'язкові «Локація» та «Тег» Обирає потрібне географічне розташування на мапі Натискає кнопку «Надіслати» Запис з'являється у відміченій точці при перегляді
Розширені сценарії	Користувач увів не всі потрібні дані, то система підсвічує потрібні для заповнення поля

Таблиця 1.4 - Варіант використання "Видалення запису"

Назва	Видалення запису
Опис	Кожен зареєстрований користувач має змогу видалити власний запис
Актор	Користувач
Передумови	Наявний профіль в системі, автентифікований користувач
Постумови	Запис видалено з системи
Сценарій	Користувач переходить на сторінку з мапою Обирає потрібну географічну точку на мапі та відкриває список записів у точці Натискає кнопку «Х» на записі, що хоче видалити Запис зникає з точки
Розширені сценарії	Користувач намагається видалити чужий запис, але не буде мати можливості натиснути «Х»

Таблиця 1.5 - Варіант використання "Редагування профілю"

Назва	Редагування профілю
Опис	Кожен зареєстрований користувач має змогу редагувати власну інформацію
Актор	Користувач
Передумови	Наявний профіль в системі, автентифікований користувач
Постумови	Потрібні поля профілю змінені
Сценарій	Користувач переходить на сторінку профілю Змінює потрібні поля Натискає кнопку «Зберегти» Поля з інформацією оновляться у бд
Розширені сценарії	Користувач увів не всі потрібні дані, то система підсвічує потрібні для заповнення поля

Таблиця 1.6 - Варіант використання "Завантаження фотографії профілю"

Назва	Завантаження фотографії профілю
Опис	Кожен зареєстрований користувач має змогу завантажити фотографію у профіль
Актор	Користувач
Передумови	Наявний профіль в системі, автентифікований користувач
Постумови	Фотографія завантажена на сервер
Сценарій	Користувач переходить на сторінку профілю Натискає на кнопку завантаження фотографії Обирає потрібну фотографію на пристрої Натискає «Зберегти» Фотографія зберігається в системі
Розширені сценарії	Користувач відміняє завантаження, в профілі залишається стара фотографія або порожня

Таблиця 1.7 - Варіант використання "Вподобання запису"

Назва	Вподобання запису
Опис	Кожен зареєстрований користувач має змогу вподобати існуючий запис
Актор	Користувач
Передумови	Наявний профіль в системі, автентифікований користувач, запис не має бути вподобаним
Постумови	Лічильник вподобань збільшено на 1
Сценарій	Користувач переходить на сторінку мапи Обирає потрібну географічну точку із записами Натискає кнопку у вигляді серця До запису додається вподобання користувача

Таблиця 1.8 - Варіант використання "Зняття вподобання із запису"

Назва	Зняття вподобання із запису
Опис	Кожен зареєстрований користувач має змогу зняти вподобання із раніше уподобаного запису
Актор	Користувач
Передумови	Наявний профіль в системі, автентифікований користувач, запис має бути вподобаним
Постумови	Лічильник вподобань зменшено на 1
Сценарій	Користувач переходить на сторінку мапи Обирає потрібну географічну точку із записами Натискає кнопку у вигляді серця Із запису знімається вподобання користувача

Таблиця 1.9 - Варіант використання "Перегляд статистики записів"

Назва	Перегляд статистики записів
Опис	Кожен зареєстрований користувач має змогу переглянути статистику активності власних записів
Актор	Користувач
Передумови	Наявний профіль в системі, Автентифікований користувач, користувач мусить мати наявні записи у системі
Постумови	-
Сценарій	Користувач переходить на сторінку власних записів Там він може споглядати інформацію про його записи з усіма деталями, а також статистикою у вигляді переглядів записів

Таблиця 1.10 - Варіант використання "Перегляд існуючих записів на мапі"

Назва	Перегляд існуючих записів на мапі
Опис	Кожен користувач має змогу переглянути існуючі записи у системі
Актор	Користувач, Анонім
Передумови	Наявні записи у системі
Постумови	-
Сценарій	Користувач переходить на сторінку мапи Знаходить цікаву йому географічну точку із записами Обирає точку для перегляду Споглядає на існуючі пости у цій точці

Узагальнена схема структурна усіх можливих варіантів використання інформаційної системи представлено на Рисунку 1.1.

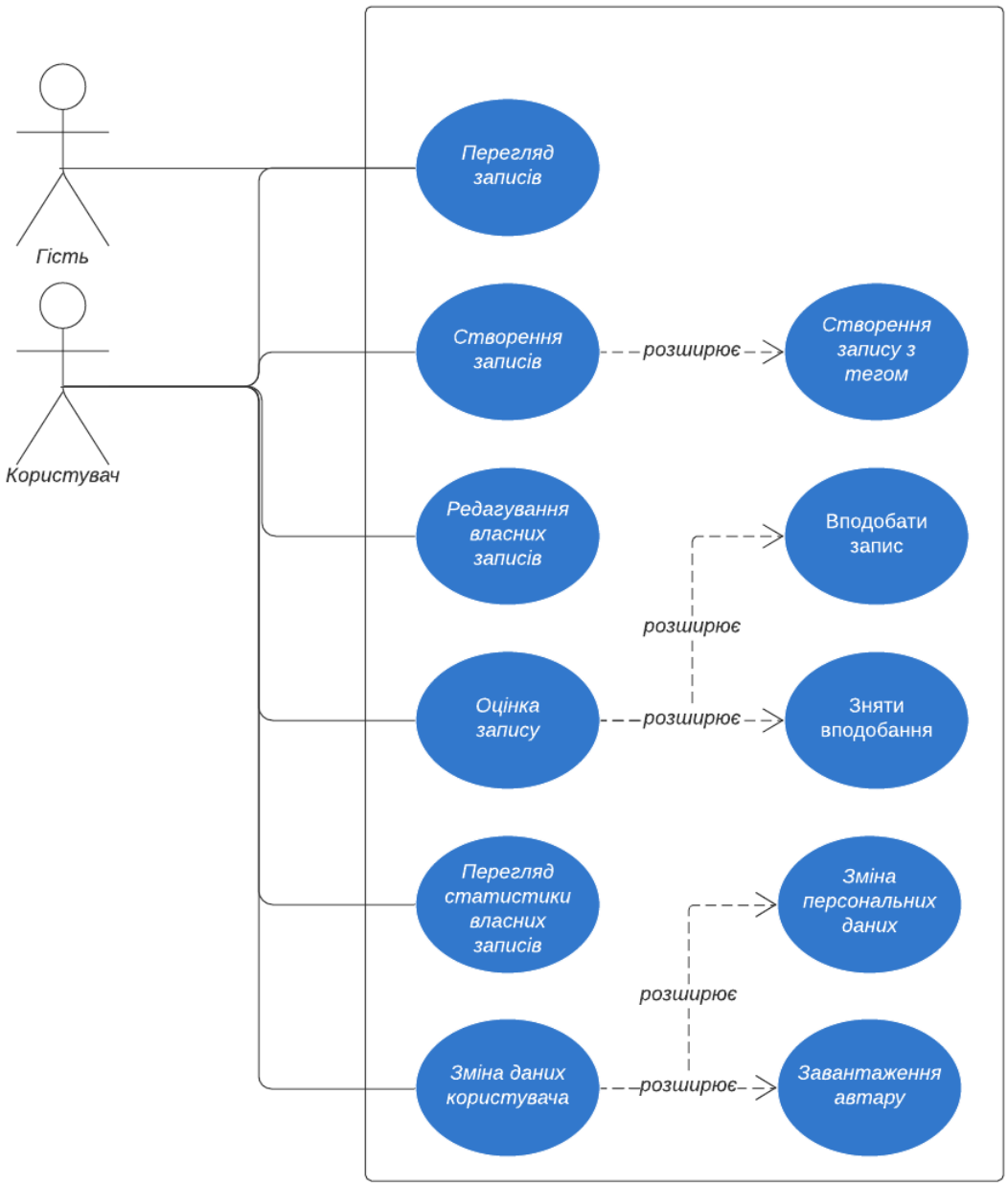


Рисунок 1.1 - Схема структурна варіантів використання

1.4.2 Розроблення нефункціональних вимог

До нефункціональних вимог відносяться:

- головна мова веб-сервісу – українська;
- працездатність в найпоширеніших веб-браузерах як Opera, Google Chrome, Mozilla Firefox, Microsoft Edge, Safari;

- збереження даних сесії для використання у наступних;
- зберігання паролів у базі даних в стані не придатному для використання третіми особами;
- веб-сервіс мусить бути створений як односторінковий застосунок для підвищення швидкодії;
- серверна частина мусить бути універсальною та може бути використана будь-яким клієнтом в незалежності від платформи.

1.4.3 Постановка комплексу завдань модулю

Описаний веб-сервіс має на меті створення єдиної платформи для обміну інформацією та досвідом відносно різних місць дозвілля.

Мета створення даної роботи – автоматизація обміну та пошуку інформації про різні місця дозвілля у розрізі геолокації та їх змісту або ж суттєвих ознак.

Для того, щоб мета вважалась досягнутою, проєкт мусить відповідати ряду технічних задач, а саме:

- реєстрація користувача;
- авторизація користувача;
- зміна даних профілю користувача;
- створення власних записів користувача про місця дозвілля;
- видалення власних записів користувача про місця дозвілля;
- редагування власних записів користувача про місця дозвілля;
- перегляд існуючих записів про місця дозвілля;
- перегляд статистики активності по власним записам користувача.

З заходів безпеки мусить бути присутнє:

- використання з’єднання з SSL;
- зберігання паролів у захешованому криптографічною функцією стані.

1.5 Висновки по розділу

Зі зростанням фінансових можливостей суспільства та їх побажань до нових та кращих стандартів відпочинку, з’явилась потреба у високоспеціалізованих сервісах для пошуку та надання рекомендацій про місця дозвілля особливо у розрізі певної геолокації.

Наявні системи мають певні недоліки зі сторони гнучкості пошуку місць дозвілля та затяжний процес поширення інформації про тимчасові або маловідомі місця та їх події.

Тому розробка нової інформаційної системи пошуку дозволить вирішити низку існуючих проблем та недоліків з метою полегшення отримання користувачем нової або ж необхідної інформації про місця бажаного дозвілля.

У цьому розділі також було охоплено актуальність та мету розробки, опис предметного середовища, наведено існуючи сервіси зі схожими функціоналам та проаналізовано відомі існуючі рішення на проблеми які вони вирішують та сторони, що могли би бути удосконаленими.

2 МОДЕЛЮВАННЯ ТА КОНСТРУЮВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

2.1 Моделювання та аналіз програмного забезпечення

З метою створення високоякісної системи надання рекомендацій для початку необхідно описати процеси взаємодії користувача з програмним забезпеченням.

Кроки створення рекомендацій наведено на рисунку 2.1. При створенні рекомендації користувач муситиме ввести необхідні дані у вигляді географічного розташування об’єкту про який пишеться відгук, коментар про місце або заклад, вказати чи рекомендується для відвідування. За бажання додаються змістовні теги за якими потім здійснюється пошук цікавих об’єктів. Після успішного додавання рекомендації вона буде відображена усім іншим користувачам. В подальшому користувач зможе видалити власний запис.

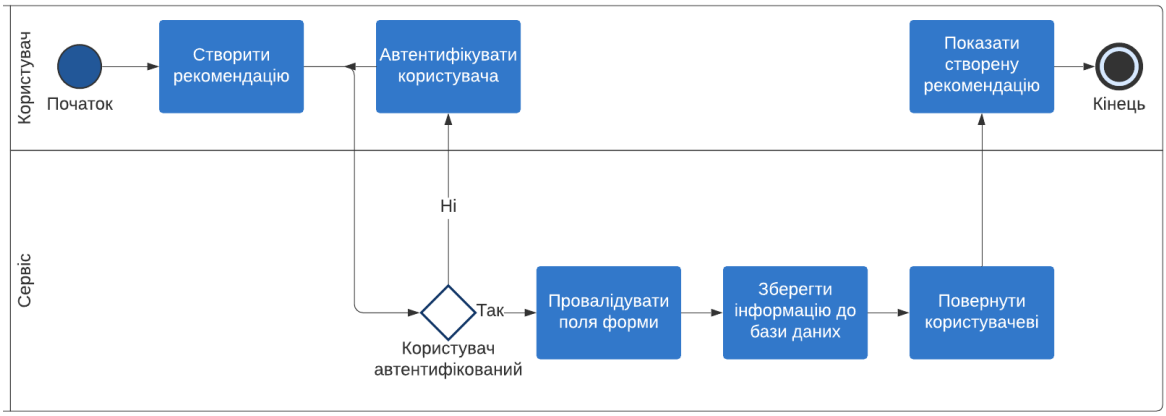


Рисунок 2.1 – Схема бізнес-процесу створення рекомендації

Одною з головних можливостей сервісу є отримання персональних рекомендацій стосовно місця для відвідування. Цей процес описано у схемі на рисунку 2.2 та передбачає від користувача лише зачекати доки виконуються необхідні обчислення. Після того на екрані з’явиться перелік місць із типом та інформацією про географічне розташування. При зацікавлені наявна можливість обрати потрібний заклад та переглянути його на карті, а також рекомендації, якщо такі існують. Після декількох обраних персональних рекомендацій для

користувача починає генеруватися особистий рейтинг уподобань за яким в подальшому відбувається отримання нових пропозицій.

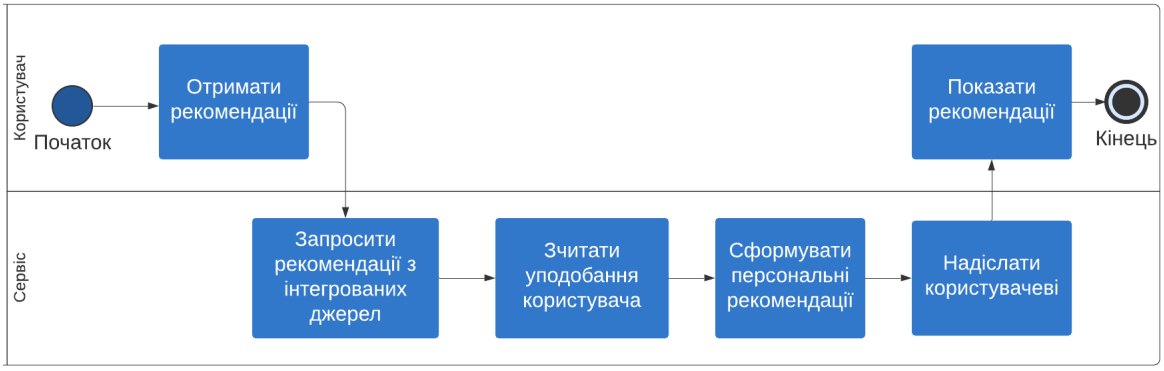


Рисунок 2.2 – Схема бізнес-процесу отримання персональних рекомендацій

Персональні рекомендації користувача генеруються на базі списку збережених варіантів вибору з запропонованих місць. Оскільки заклади поділяються за типом, то формування рейтингу пріоритетів відбувається саме за ними. Подальша фільтрація та сортування запропонованих місця опирається на особистий список користувача, котрий оновлюється після кожного нового вибору.

Також користувач має змогу змінити власні налаштування на сторінці профілю. Програмно передбачено зміну:

- Ім’я;
- емейл;
- логін;
- пароль;
- телефон;
- стать.

При зміні полів форми відбувається початкова перевірка на клієнтській частині, якщо усі поля заповнені, то надсилається запит на сервер. Наступним кроком перевіряється чи є можливим зміна інформації на введені дані, а саме чи є унікальним «Логін», «Емейл» та «Телефон». Якщо буде невірно введено хоча

б одне поле, то зміна не відбудеться та користувачеві буду повернута помилка із вказівкою на проблему.

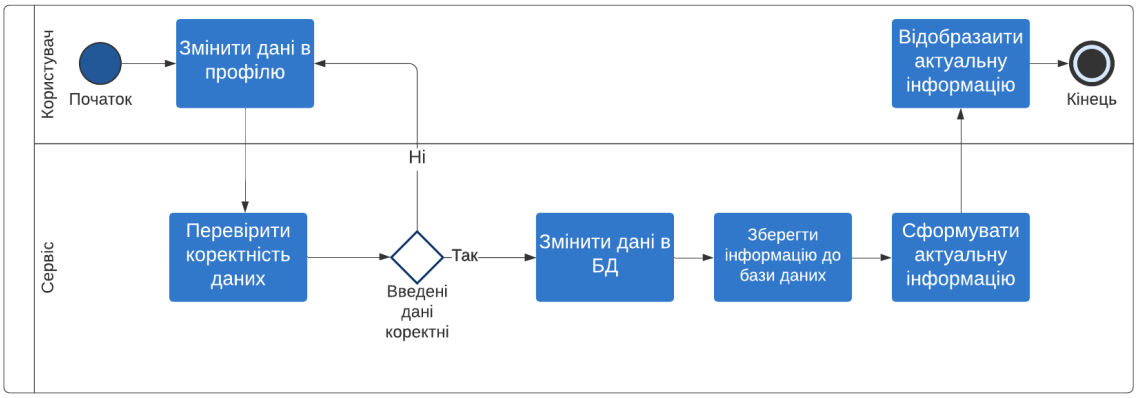


Рисунок 2.3 – Схема бізнес-процесу зміни налаштувань профілю

Для інтерфейсу користувача розроблено прототипи екранів за допомогою спеціалізованого інструменту – Figma. Під час розробки деякі екрани були змінені задля покращення зручності використання сервісу та більш сучасного виду.

На рисунку 2.4 показано вигляд сторінки входу. В ній наявна навігація до екрану реєстрації для нових користувачів.

Рисунок 2.4 – Екран входу до системи

На рисунку 2.5 продемонстровано вигляд сторінки реєстрації. Після проходження етапу створення нового профілю, користувач автоматично автентифікується у системі.

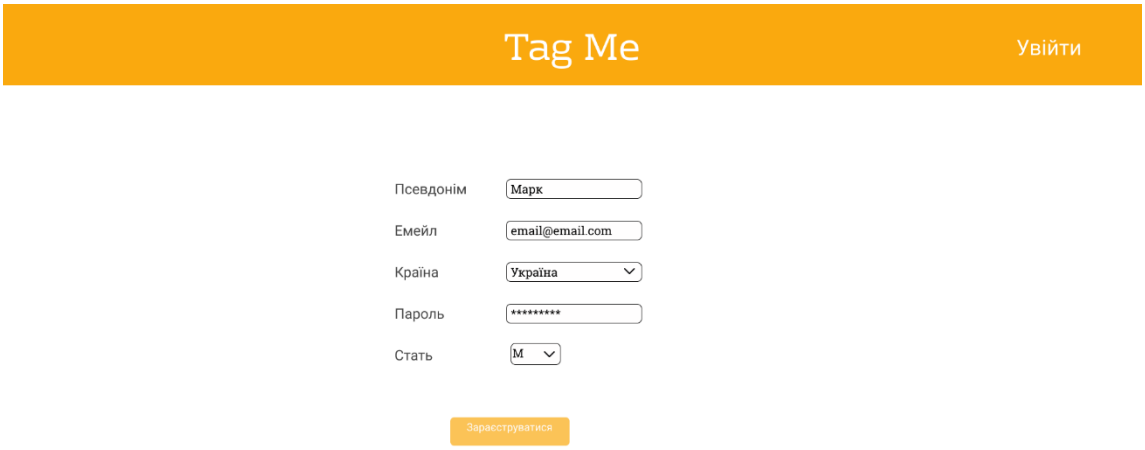


Рисунок 2.5 – Екран реєстрації у системі

Для автентифікованого користувача головним екраном є домашня сторінка із зображеною мапою та списком рекомендацій поруч з нею. Після входу до систему з’являється навігація по сервісу у лівій частині. На головній сторінці кожен користувач має змогу створити власну рекомендацію та переглянути існуючі на мапі.

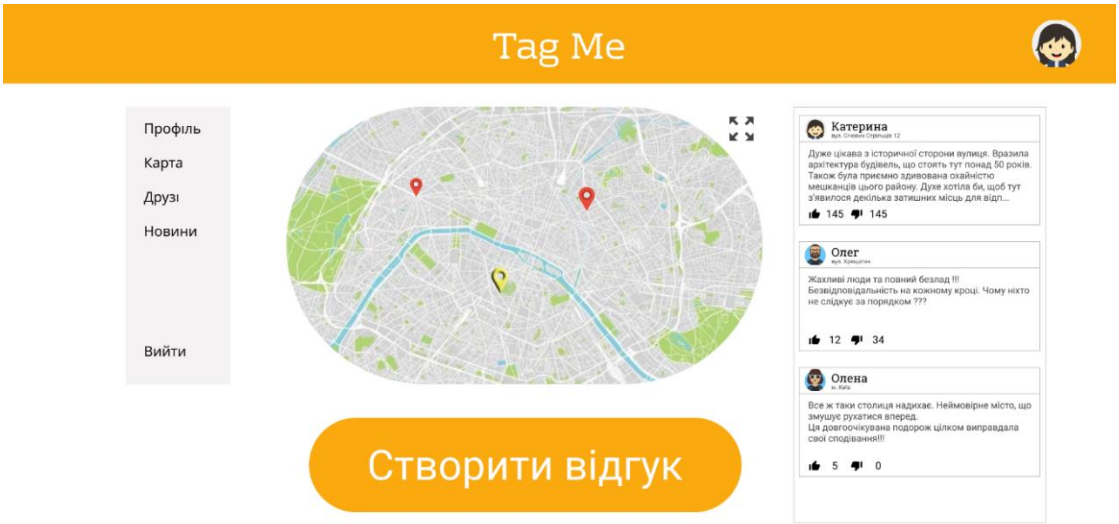


Рисунок 2.6 – Головний екран сервісу

На сторінці профілю Рисунок 2.7 користувач може переглянути власну інформацію, фотографію, а також змінити їх за бажанням.

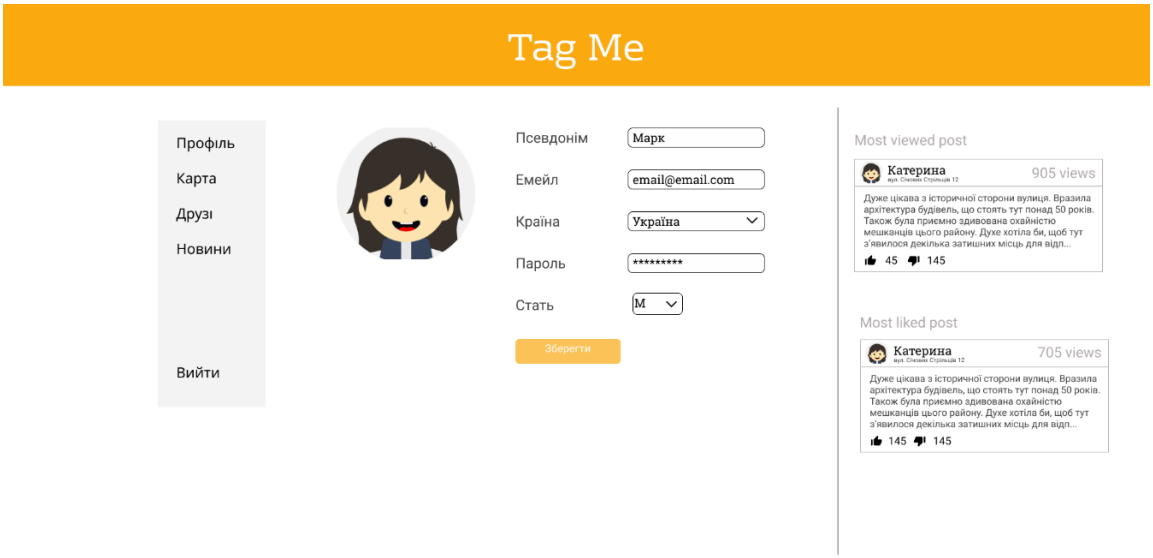


Рисунок 2.7 – Екран профілю користувача

Початкові прототипи слугують для чіткого уявлення на етапах моделювання та проектування. Вихідне програмне забезпечення не зобов'язане чітко відповідати кожному рисунку, та може бути модифіковано з метою отримання більш якісної інформаційної системи.

2.2 Архітектура програмного забезпечення

При розробці системи рекомендацій було використано широко розповсюджений підхід розбиття програмного забезпечення на частини з обмеженою відповідальністю за певну логіку, а саме трирівнева архітектура з клієнтською частиною, серверною та рівнем бази даних.

У кожній з частин були використані відповідні інструменти, що мають готові та зручні рішення для розробки. Так на рівні бази даних було використано MS SQL Server для збереження інформації та даних, для серверної обробки інформації та реалізації логіки застосування було обрано мову програмування C# на платформі .NET Core, що може бути розгорнута на будь-якій операційній системі, а для створення інтерфейсу та взаємодії з користувачем взято Angular,

який є JavaScript фреймворком, та стилістично оформлено за допомогою CSS бібліотеки UIKit.

Рисунок 2.8 – Архітектура програмного забезпечення

У кожній з частин були використані відповідні інструменти, що мають готові та зручні рішення для розробки. Так на рівні бази даних було використано MS SQL Server для збереження інформації та даних, для серверної обробки інформації та реалізації логіки застосування було обрано мову програмування C#

на платформі .NET Core, що може бути розгорнута на будь-якій операційній системі, а для створення інтерфейсу та взаємодії з користувачем взято Angular, який є JavaScript фреймворком, та стилістично оформлено за допомогою CSS бібліотеки UIKit.

При розробці серверної частини було враховано провідні підходи у написанні коду та розмежовано відповідальність за рівнями відповідно до багатошарової моделі побудови програмного забезпечення. Керуючись ідеями предметно-орієнтованого проектування основним та незалежним від інших проектом було обрано збірку, що включає усі базові моделі та логіку взаємодії з ними. Наступним рівнем став проєкт з функціональною логікою системи до якого входять усі сервіси предметної області, які займаються обробкою запитів користувачів та діють за наведеними раніше схемами. Потім слідує рівень взаємодії з базою даних, де описується структура таблиць та методи доступу до даних. Ця частина також включає в себе реалізацію шаблону «Репозиторій» для кожного типу доменної області. До купи збирає усі рівні веб-сервіс та надає точку доступу для взаємодії з програмним забезпеченням у мережі Інтернет. Додатковим шаром є проєкт з тестами, що має на меті підтримку якості та коректності роботи застосування.

Така структура дозволяє розділяти код за логічними сферами відповідальності та підтримувати гнучкість у здатності масштабування. Кожна з частин з частин може бути змінена ізольовано від інших або ж бути заміненою на іншу у випадку необхідності.

Рисунок 2.9 – Архітектура серверної кодової бази

Структура бази даних графічно наведена на рисунку 2.4 разом із типом відносин між таблицями. Схема знаходиться у третій нормальній формі та не має надлишковості даних.

Таблиці пов’язані між собою в більшості один-до-багатьох, але наявне одне з’єднання багато-до-багатьох між тегами та постами з метою можливості позначати багатофункціональні місця. Для цього використовується таблиця «PostTags».

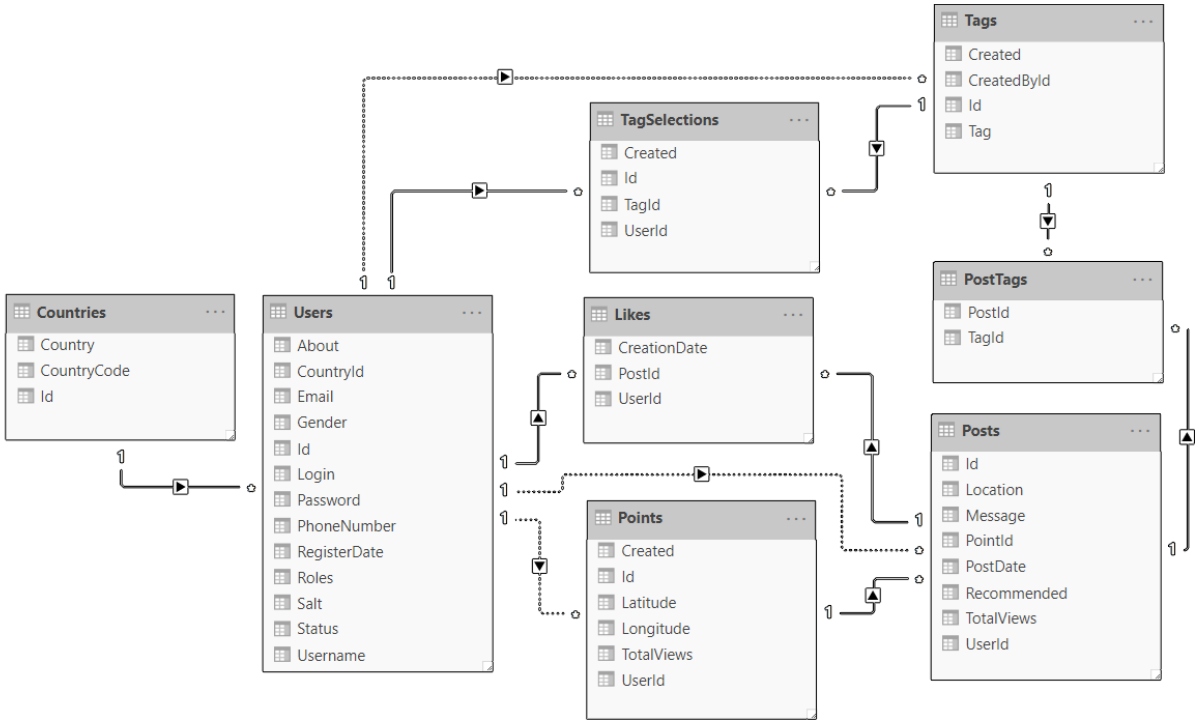


Рисунок 2.10 – Схема бази даних

Всього схема містить 8 таблиць, що зберігають інформацію про користувача, його записи, статистику взаємодії, а також теги та географічні точки. Детальний опис кожної таблиці наведено у таблицях 2.1 - 2.8

Таблиця 2.1 - Інформація про таблицю «Posts»

Назва атрибуту	**Опис**	**Тип даних**	**Обов'язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**
Id	Ідентифікатор запису	Bigint	+	+	+	-
Message	Текстове повідомлення	Nvarchar(1000)	+	-	-	-

Продовження таблиці 2.1

Recommended	Позначка рекомендації місця	Bit	+	-	-	-
Location	Фізична адреса місця	Nvarchar(100)	+	-	-	-
PostDate	Дата та час створення запису	Datetime2(7)	+	-	-	-
UserId	Ідентифікатор користувача, що створив запис	Bigint	+	-	-	+
PointId	Ідентифікатор точки до якої прив’язаний запис	Bigint	+	-	-	+
TotalViews	Перегляди запису	Bigint	+	-	-	-

Таблиця 2.2 – Інформація про таблицю «Points»

Назва атрибуту	**Опис**	**Тип даних**	**Обов’язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**
Id	Ідентифікатор точки	Bigint	+	+	+	-
Latitude	Широта розташування точки	Float	+	-	-	-

Продовження таблиці 2.2

Longitude	Довгота розташування точки	float	+	-	-	-
Created	Дата та час створення точки	Datetime2(7)	+	-	-	-
UserId	Ідентифікатор користувача, що створив точку	Bigint	+	-	-	+
TotalViews	Перегляди точки	Bigint	+	-	-	-

Таблиця 2.3 – Інформація про таблицю «Users»

Назва атрибуту	**Опис**	**Тип даних**	**Обов' язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**
Id	Ідентифікатор користувача	Bigint	+	+	+	-
Username	Ім'я користувача	Nvarchar(100)	+	-	-	-
Gender	Позначка статі	Bit	-	-	-	-
Email	Емейл користувача	Nvarchar(300)	-	+	-	-
Login	Логін для входу	Nvarchar(100)	+	+	-	-
Password	Хеш від паролю	Nvarchar(500)	+	-	-	+
Salt	Перегляди запису	Nvarchar(500)	+	-	-	-

Продовження таблиці 2.3

PhoneNumber	Номер телефону	Nvarchar(20)	-	+	-	-
CountryId	Ідентифікатор країни	Bigint	+	-	-	+
RegisterDate	Дата та час реєстрації	Datetime2(7)	+	-	-	-

Таблиця 2.4 – Інформація про таблицю «PostTags»

Назва атрибуту	**Опис**	**Тип даних**	**Обов’язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**
PostId	Ідентифікатор посту	Bigint	+	-	-	+
TagId	Ідентифікатор тегу	Bigint	+	-	-	+

Таблиця 2.5 – Інформація про таблицю «Likes»

Назва атрибуту	**Опис**	**Тип даних**	**Обов’язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**
UserId	Ідентифікатор користувача, котрий вподобав	Bigint	+	-	-	+

Продовження таблиці 2.5

PostId	Ідентифікатор посту, що був вподобаний	Bigint	+	-	-	+
CreationDate	Час уподобання	Datetime2(7)	+	-	-	-

Таблиця 2.6 – Інформація про таблицю «Tags»

Назва атрибуту	**Опис**	**Тип даних**	**Обов' язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**
Id	Ідентифікатор тегу	Bigint	+	+	+	-
Tag	Назва тегу	Nvarchar(128)	+	+	-	-
Created	Час створення тегу	Datetime2(7)	+	-	-	-
CreatedById	Ідентифікатор користувача, котрий створив тег	Bigint	+	-	-	+

Таблиця 2.7 – Інформація про таблицю «Countries»

Назва атрибуту	**Опис**	**Тип даних**	**Обов' язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**

Продовження таблиці 2.7

Id	Ідентифікатор країни	Bigint	+	+	+	-
Country	Назва країни	Nvarchar(128)	+	+	-	-
CountryCode	Код країни у міжнародному форматі	Nvarchar(3)	+	+	-	-

Таблиця 2.8 – Інформація про таблицю «TagSelections»

Назва атрибуту	**Опис**	**Тип даних**	**Обов’язкове**	**Унікальне**	**Первинний ключ**	**Зовнішній ключ**
Id	Ідентифікатор вибору тегу	Bigint	+	+	+	-
TagId	Обраний тег	Bigint	+	-	-	+
Created	Код країни у міжнародному форматі	Datetime2(7)	+	-	-	-
UserId	Ідентифікатор користувача, котрий обрав тег	Bigint	+	-	-	+

2.3 Конструювання програмного забезпечення

Для основних сервісів програмного забезпечення побудовано схему структурну класів з визначеною функціональністю. Опис сервісів наведено на рисунку 2.7.

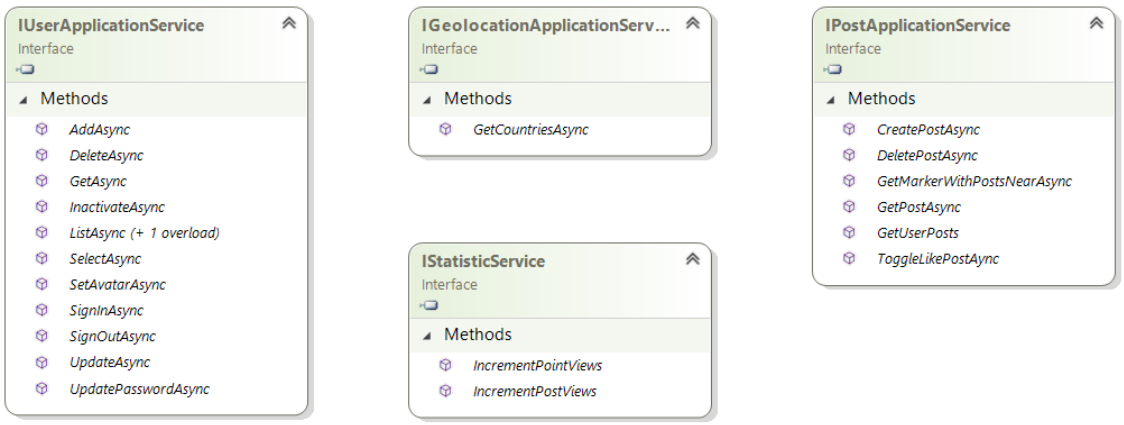


Рисунок 2.11 – Схема структурна класів сервісів

Схему структурну класів для предметної області програмного забезпечення наведено на Рисунку 2.7. На ньому наведено класи з яких утворюється рекомендація. До неї входить географічна точка, користувач, що її створив, теги, що за нею закріплені та вподобання. Додатковими класами для більш зручного користування системою є фотографію профілю користувача та країна перебування.

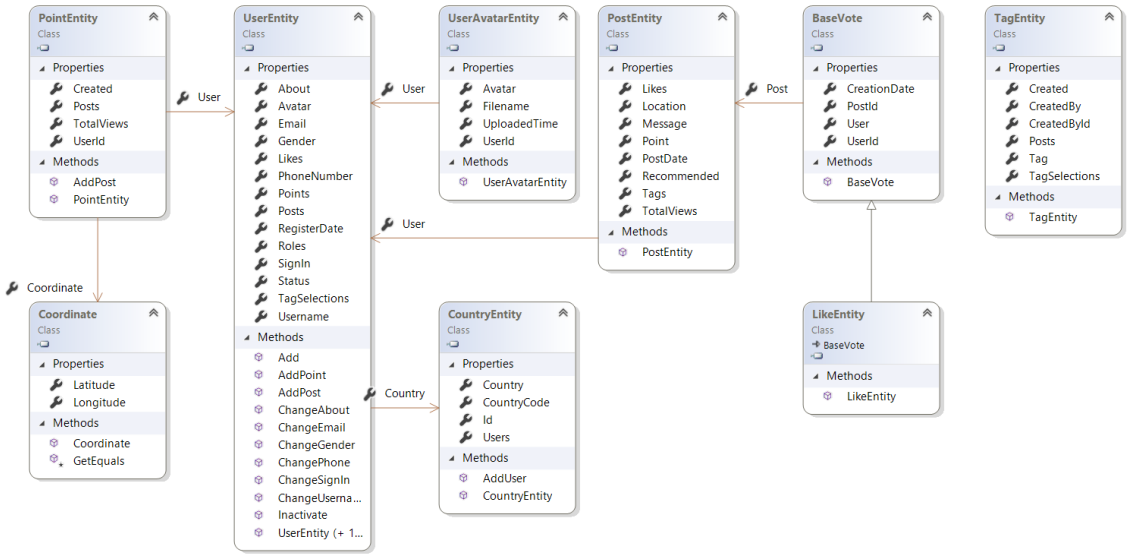


Рисунок 2.12 – Схема структурна класів предметної області

Сервіси як частина забезпечення з логікою предметної області відокремлена від бази даних абстракцією у вигляді шаблону проектування

«Репозиторій», що дозволяє розділити обов'язки різних частин системи. Для класів-абстракцій також побудовано діаграму класів.

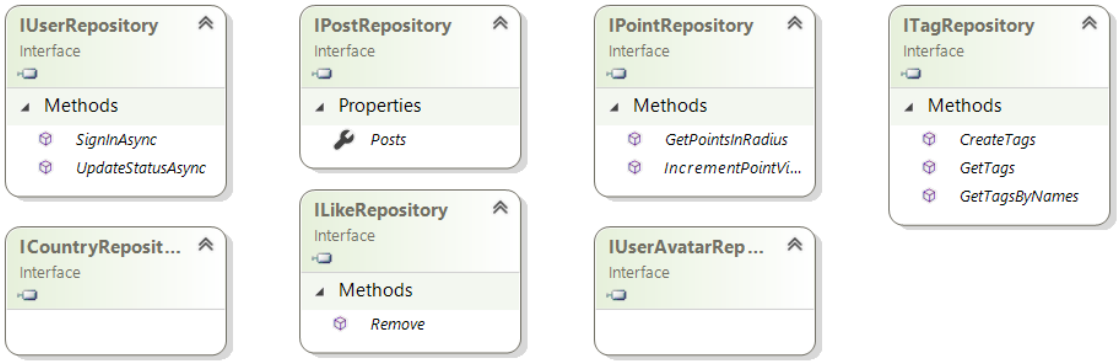


Рисунок 2.13 – Схема структурна класів репозиторіїв

Узагальнена схема структурна класів архітектурного підходу наведена на Рисунку 2.10. Зліва на право розташовані сутності, що взаємодіють один з одним. Так основним є об'єкт предметної області в який наповнюється даними із бази даних. У свою чергу сервіс, використовуючи репозиторій, отримує дані незалежно від їх постачальника, виконує необхідні операції та перетворення, отриманий результат передається до контролера, що формує відповідь на запит користувача.

Архітектурний шаблон проектування «Одиниця роботи» використовується для забезпечення виконання потрібних операцій у вигляді транзакції на рівні сервісів.

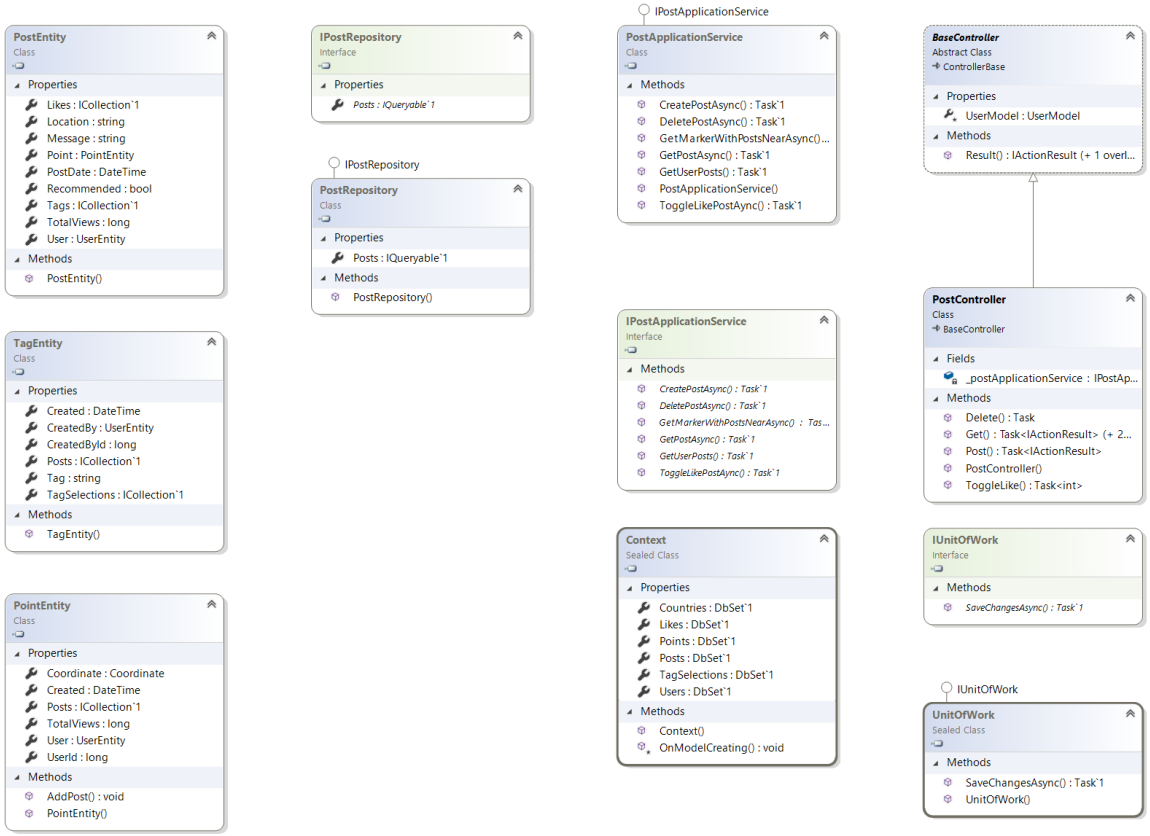


Рисунок 2.14 – Схема структурна класів архітектурного підходу

Детальна інформація стосовно класів предметної області та їх сферу відповідальності наведена у таблиці 2.9.

Таблиця 2.9 – Таблиця опису класів предметної області

Клас	Опис
CountryEntity	Описує структуру об’єкту «Країна». Має поля ідентифікатору, назви країни та міжнародного коду країни.
PointEntity	Клас географічної точки, до якої прикріплюються записи користувачів. Описується географічною довготою та широтою. Має унікальний ідентифікатор та посилання на користувача який створив.

Продовження таблиці 2.9

PostEntity	Містить інформацію про запис, а саме рекомендацію від користувача, текстове повідомлення, дату публікації, фізичну адресу, що може бути додана за бажанням
TagEntity	Описує тег та включає інформацію про дату створення та особу, яка ініціювала
UserEntity	Об’єкт користувача, що містить введені дані профілю та інформацію потрібну для автентифікації
UserAvatarEntity	Сутність для збереження фотографії профілю у вигляді масиву байт
TagSelectionEntity	Цей клас описує вибір типу закладу з отриманих користувачем персональних рекомендацій. Відіграє важливу роль у алгоритмі формування нових пропозиція для користувача
PostTagEntity	Сутність, що виконує роль сполучника у зв’язку багато до багатьох між записами користувачів та тегами.
LikeEntity	Клас об’єкту уподобання, що включає у себе уподобаний пост та користувача, котрий ініціював.

Перелік та опис головних сервісів, що містять логіку виконання, перевірки та оброки інформації наведено у таблиці 2.10.

Таблиця 2.10 – Таблиця опису головних сервісів ПЗ

Назва сервісу	**Опис**
GeolocationApplicationService	Відповідає за обробку інформації пов’язаної з географічним розташуванням

Продовження таблиці 2.10

UserApplicationService	Займається обробкою інформації про користувача та даних що відносяться до профілю
StatisticService	Сервіс обробляє статистичні дані відносно переглядів, уподобань користувачів
PostApplicationService	Працює із записами від користувачів. А саме їх додавання, видалення, перегляд, також обробляє інформацію стосовно географічних точок

Перелік та опис Web API точок доступу до програмного забезпечення перелічено у таблиці 2.11.

Таблиця 2.11 – Таблиця точок доступу до контролера «Users»

Назва	**HTTP метод**	**Вхідні параметри**	**Формат відповіді**	**Опис**
Register	POST	User newUser	User – профіль новоствореного користувача	Метод для реєстрації нових користувачів у системі
Profile	GET	-	User - Інформація про профіль користувача	Використовується для отримання інформації про автентифікованого користувача. Повертає об’єкт користувача разом з фото

Продовження таблиці 2.11

SetAvatar	POST	Byte[] avatar	Byte[] – фотографія	Метод, що дозволяє завантажити власну фотографію в профіль, яка потім буде відображена іншим користувачам системи
SignIn	POST	String login String password	При успішній автентифікації – токен При невдалій – помилку	Потрібен для можливості автентифікації користувача у системі
SignOut	POST	-	-	Викликається при виході із системи, а саме при очищені користувачем токену автентифікації
Update	POST	User updatedUser	User – об’єкт користувача після змін, або ж помилку спроби зміни даних	Використовується для зміни даних профілю

Продовження таблиці 2.11

ChangePassword	POST	String oldPassword String newPassword	Error – повідомлення про помилку у випадку неправильних вхідних даних	Окремий метод зміни інформації профілю – паролю

З метою відстежування активності користувачів, їх зацікавленість та персональні характеристики було створено електронний звіт за допомогою системи Power BI.

Аналітична система широко розповсюджена у сфері аналізу даних та має вбудований функціонал для роботи з джерелами інформації, у випадку цієї роботи – база даних MS SQL Server. Після зчитування інформації можна відредагувати її до більш зрозумілої людині форми з нормалізованого стану.

Цікавими моментами для вивчення та аналізу користувачів є їх активність та інтерес, тому більшість діаграм та графіків спрямовані на відображення статистики взаємодії користувачів з програмним забезпеченням.

Звіти інформаційної системи включають в себе розподіл користувачів за статтю та країнами перебування, відношення рекомендованих місць до не рекомендованих, рейтинг найбільш вживаних тегів, розподіл часу створення рекомендацій за проміжками впродовж доби та днями тижня, також існує звіт у вигляді теплової мапи з позначенням місць найбільшого скупчення створених рекомендацій за географічним розташуванням.

Оскільки звіти відображають інформацію пов’язану між собою з одного джерела даних, то вони є інтерактивними та можуть взаємодіяти між собою. Це означає, що виділивши цікавий критерій з певної діаграми, інші графіки відфільтруються або трансформуються під обраний фактор.

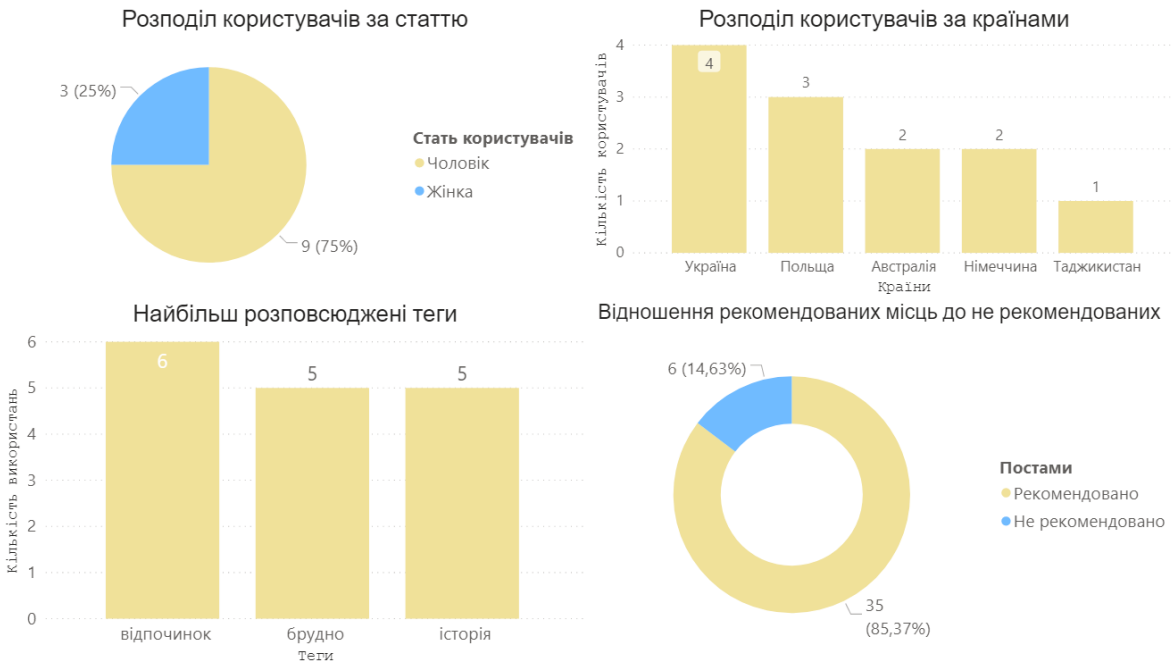


Рисунок 2.15 – Звітність інформаційної системи

Power BI напряму під'єднаний до бази даних, тому уся звітність може бути оновлена актуальними даними у будь-який час. Представлення даних та статистики можуть бути відфільтровані за критерієм, відсортовані у порядку спадання або зростання за обраними значеннями, а також графіки та способи демонстрації можна швидко замінити на інші, що будуть більш наочніше демонструвати ту чи іншу динаміку.

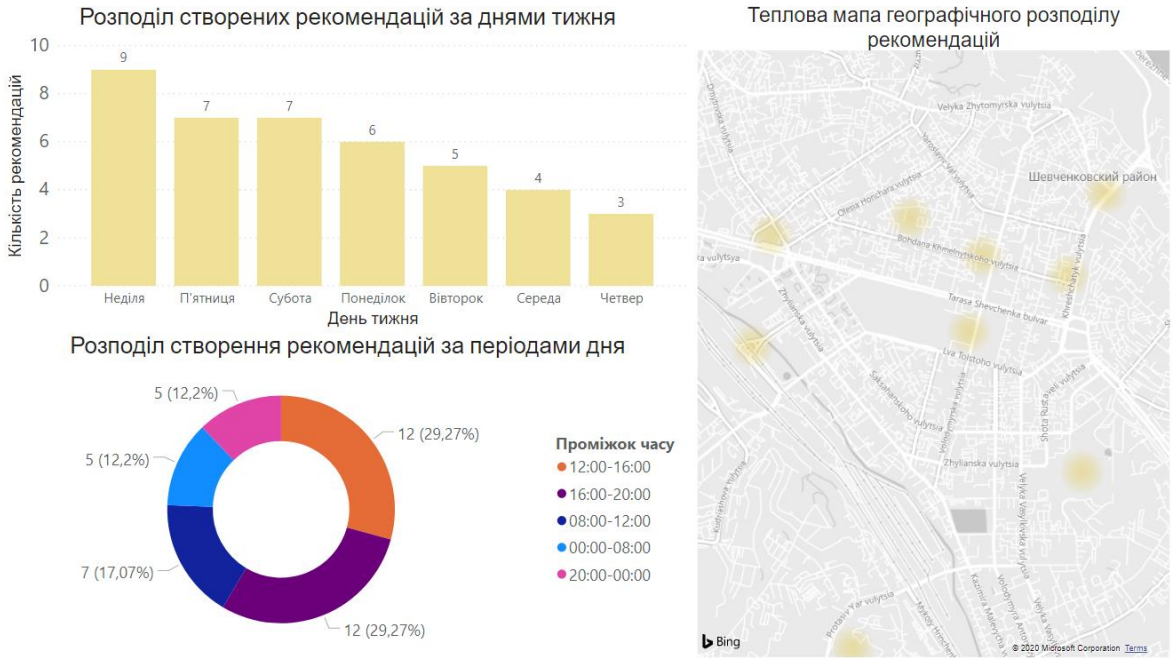


Рисунок 2.16 – Звітність інформаційної системи

На основі інформації, що демонструється у звітах, можна зрозуміти актуальність та популярність частин інформаційної системи, щоб сформувати пріоритети для нової функціональності та робити висновки щодо середнього портрету користувача.

2.4 Аналіз безпеки даних

При проектуванні слід враховувати ряд небезпечних факторів, що можуть суттєво погіршити або ж навіть змінити логіку виконання програми у несприятливий для користувачів напрям.

Для запобігання XSS атак при зчитуванні вхідних даних з клієнтської частини на сервері відфільтровуються частини, що вміщують html розмітку, а також скрипти, щоб при відображені на веб сторінці введена інформація ніяким чином не впливала на вміст сервісу та його дію.

Пряма взаємодія користувача з БД через SQL ін’єкцію не є можливою, оскільки усі місця дані від користувача, що використовуються у запитах підставляються виключно у вигляді параметрів, а отже залишаються незмінними.

Уся комунікація користувача із сервером цілком побудована на безпечному HTTP з’єднані завдяки SSL протоколу. Таким чином шифрується важлива інформація від користувача та дає впевненість, що відповідь на запит надіслана саме із серверу сервісу.

Зберігання паролів не відбувається у їх справжньому вигляді. Натомість для кожного паролю генерується унікальна сіль, що додається до паролю і потім виконується хешування криптографічною функцією XSalsa20. Результат зберігається у базі даних, а при кожній потребі в автентифікації ці дії повторюються та отримане значення порівнюється зі збереженим. У такий шлях єдиним хто знає пароль залишається користувача і навіть у випадку отримання доступу до інформації в базі даних зловмисники не зможуть скористатися паролем користувача.

2.5 Висновки по розділу

У пунктах цього розділу було оглянуто технічну сторону процесу створення та проектування геоінформаційної системи. До розглянутих аспектів відносяться: модель концептуального рівня з наведеними до неї рисунками діаграм процесів, що поетапно пояснюють необхідні для виконання операції при взаємодії з системою надання рекомендацій, також описано використані технології при побудові програмного забезпечення, наведено архітектурні моделі взаємозв’язків класів у коді і відношення таблиць у базі даних, наведено низку методів спрямованих на убезпечення інформації користувачів та сервісу в цілому.

3 АНАЛІЗ ЯКОСТІ ТА ТЕСТУВАННЯ ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

3.1 Аналіз якості ПЗ

Аналіз і підтримка якості програмного забезпечення відбувається за допомогою покриття коду тестами, а також створенні тестових сценарії для можливості регресійного тестування після появи нової функціональності.

З програмного боку було створено окремий проєкт для внутрішніх юніт-тестів, які допомагають найпершими виявити зламані частини коду, оскільки можуть бути запущені одразу після змін і займають найменше часу для прогону. В першу чергу такими тестами були покриті об’єкти з предметної області та сервіси, що відповідають безпосередньо за логічну обробку даних.

До ефективного тестування відносяться інтеграційні тести, що охоплюють взаємодію між різними частинами програмного забезпечення. Таких тестів існує лише декілька, а саме на критичних шляхах у місцях, що є найбільш важливими.

Наступним кроком у підтримці якості стало складання плану ручного тестування для перевірки правильності роботи сервісу зі звичайними даними, а також випробовування системи з граничними значеннями та неправильно наданою інформацією.

План тестування програмного забезпечення складений за поширеним стандартом IEEE 828. До нього увійшов ряд функціональних частин, що належать тестуванню:

- Реєстрація нового користувача;
- спроба реєстрації вже з існуючим в системі логіном;
- автентифікація існуючого користувача;
- спроба автентифікації з невірним паролем;
- завантаження фотографії профілю;
- зміна даних профілю;
- спроба зміни на незаповнені поля;

- зміна паролю користувача;
- перегляд персональних рекомендацій;
- створення відгуку про місце;
- видалення власного відгуку;
- вподобання існуючого відгуку;
- перевірка статистики відгуків.

Для кожної з вищенаведених функціональних частин продумано та створено тест-кейс.

3.2 Опис процесів тестування

Процес тестування проводить виключно на конфігурації Release з метою досягнення найбільш схожих умов до робочого стану. Процес тестування поділений на декілька етапів.

До першого етапу відноситься прогін юніт-тестів після написання кодової бази. Вони допомагають виявити проблеми найпершими ще задовго до розгортання програмного забезпечення.

Наступним етапом є запуск сервісу та перевірка функціональності у ручному режимі, дотримуючись сценаріїв у тест кейсах. Для цього створено таблицю 3.1 з описом, кроками відтворення та очікуваним результатом для кожної функціональності, що має бути протестована.

Таблиця 3.1 – Таблиця тест кейсів

Номер	**Опис**	**Кроки відтворення**	**Очікуваний результат**
1	Успішна реєстрація нового користувача	Відкрити сторінку реєстрації. Заповнити необхідні поля форми. Логін не має бути наявним в системі Натиснути «Зареєструватися»	Створений новий користувач.

Продовження таблиці 3.1

2	Спроба зареєструвати користувача вже з існуючим у системі логіном	Відкрити сторінку реєстрації. Заповнити необхідні поля форми. Логін має співпадати з наявним у системі. Натиснути «Зареєструватися».	Показ повідомлення про помилку. Новий користувач не створений.
3	Успішна автентифікація користувача	На сторінці автентифікації заповнити поля Логіну і Паролю, що співпадають з наявними у системі. Натиснути «Увійти»	Вхід до профілю.
4	Спроба автентифікації з невірним паролем	На сторінці автентифікації заповнити поля Логіну і Паролю, що різняться від наявних у системі. Натиснути «Увійти»	Показ помилки.
5	Зміна паролю на сторінці профілю	Користувач має бути автентифікований. Відкрити вікно зміни паролю. Вести існуючий пароль та новий. Натиснути «Зберегти».	Встановлення нового паролю для профіля.
6	Спроба зміни паролю з невірно введеним існуючим	Користувач має бути автентифікований. Відкрити вікно зміни паролю. Вести пароль, що не співпадає з існуючим, та новий. Натиснути «Зберегти».	Показ повідомлення про помилку.

Продовження таблиці 3.1

7	Перегляд персональних рекомендацій	Користувач має бути автентифікований. Перейти на головну сторінку з мапою. Відкрити вікно персональних рекомендацій.	Показ місць дозвілля.
8	Створення рекомендації	Користувач має бути автентифікований. Перейти на головну сторінку з мапою. Заповнити форму рекомендації. Додати теги та розташування. Надіслати рекомендацію.	Створено нову рекомендацію у системі з обраними тегами та розташуванням.
9	Видалення власної рекомендації	Користувач має бути автентифікований. Перейти на головну сторінку з мапою. Обрати на мапі точку з власною рекомендацією. Відкрити перелік рекомендацій у точці. Натиснути та підтвердити видалення власної рекомендації.	Видалено рекомендацію з системи.

Продовження таблиці 3.1

10	Уподобання існуючої рекомендації	Користувач має бути автентифікований. Перейти на головну сторінку з мапою. Переглянути рекомендації на географічній точці. Натиснути на вподобання рекомендації.	Збільшено кількість уподобань на рекомендації.
11	Перегляд статистики активності власних рекомендацій	Користувач має бути автентифікований. Перейти на сторінку власних рекомендацій. Переглянути перегляди реокмендацій.	Показ інформації про взаємодію з рекомендацією.
12	Перегляд існуючих рекомендацій на мапі	Користувач має бути автентифікований. Мають бути наявні рекомендації в системі. Перейти на сторінку з мапою. Відкрити точку з рекомендаціями для перегляду.	Показ переліку існуючих рекомендацій у точці.
13	Завантаження фотографії профілю	Користувач має бути автентифікований. Перейти на сторінку профілю. Завантажити фотографію.	Фотографія відображається у профілі користувача.

3.3 Опис контрольного прикладу

Для описання прикладу було взято функціональність у вигляді створення власної рекомендації. Процес тестування описується такими кроками:

- реєстрація у системі. Перехід на сторінку реєстрації, заповнення необхідних полів форми, а саме Логін, Пароль, Нікнейм, Емейл, Телефон, Стать та Країна. Натиснути «Зареєструватися»;

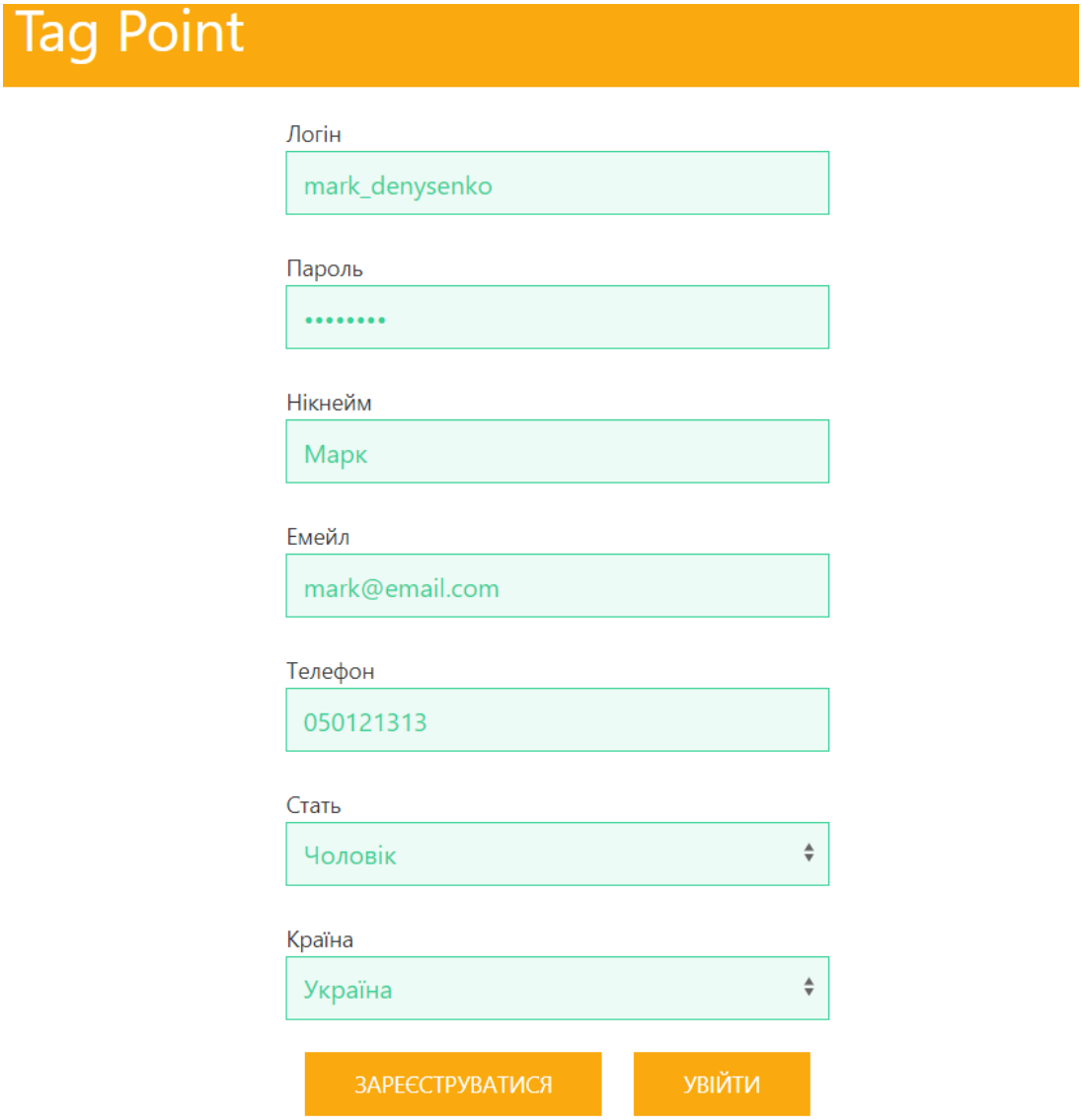


Рисунок 3.1 – Реєстрація нового користувача

- пошук місця на мапі для якого буде створено рекомендацію. Для цього можна скористатися мишкою та графічно знайти на мапі бажане місце,

натиснути в позиції місця, щоб поставити географічний маркер для майбутньої точки;

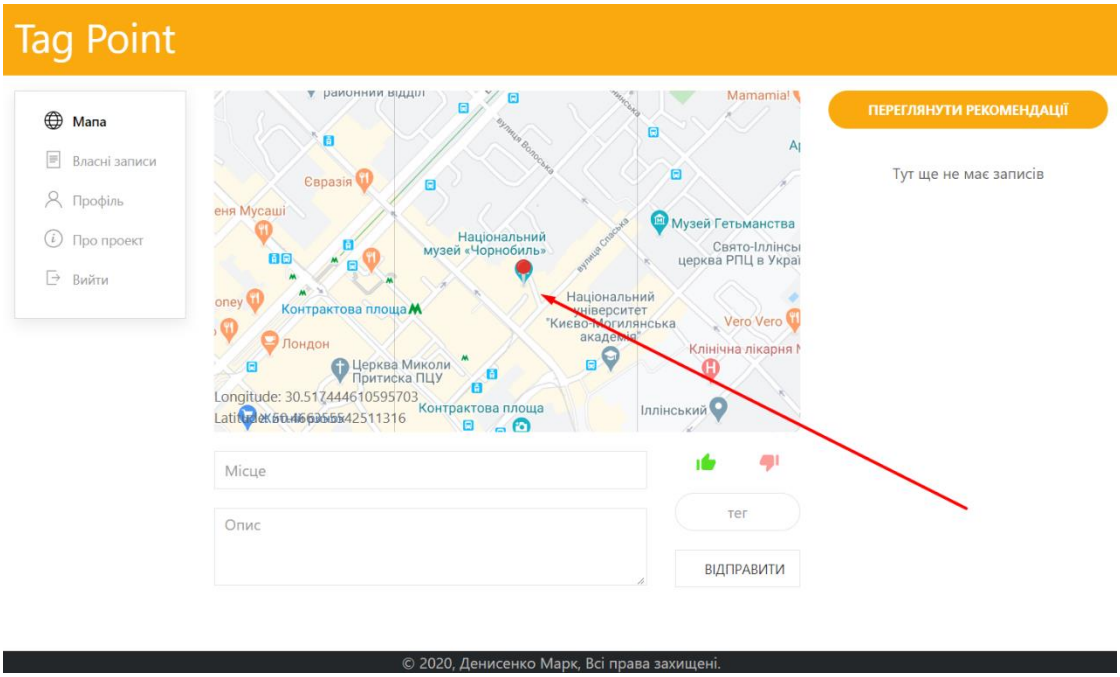


Рисунок 3.2 – Пошук місця на мапі

- заповнення фізичної адреси використовуючи запропоновані місця поруч із розташуванням маркеру. Буде наданий список із назв, де можна обрати необхідний варіант;

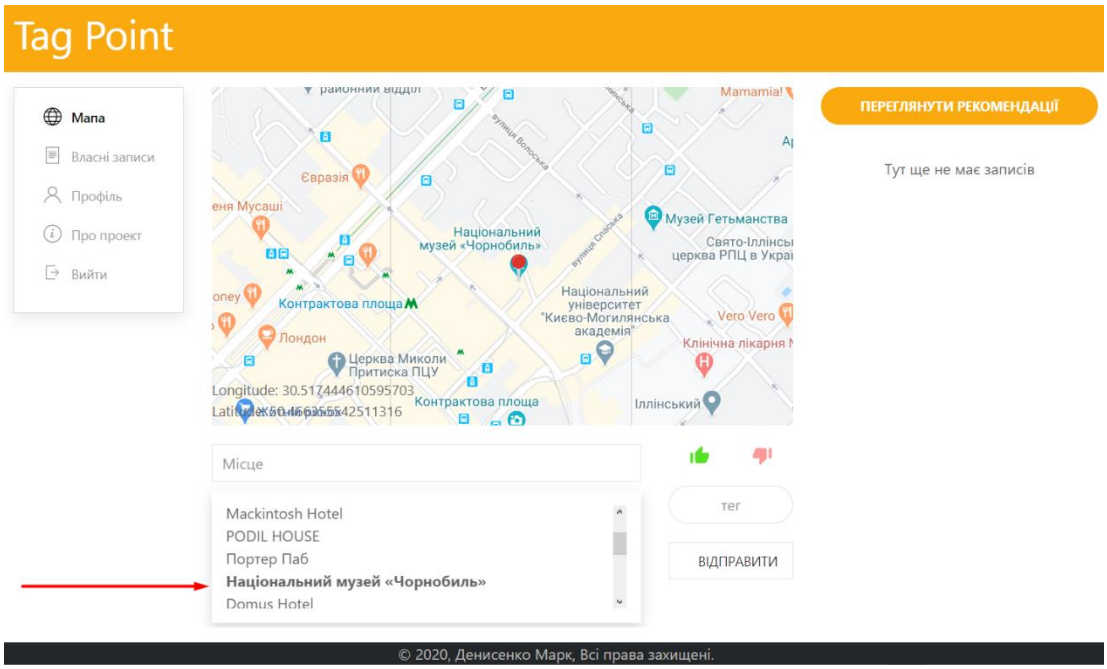


Рисунок 3.3 – Підпис розташування місця

- текстовий опис враження про місце має бути наданий у найбільше поле форми та має обмеження в 1000 символів;

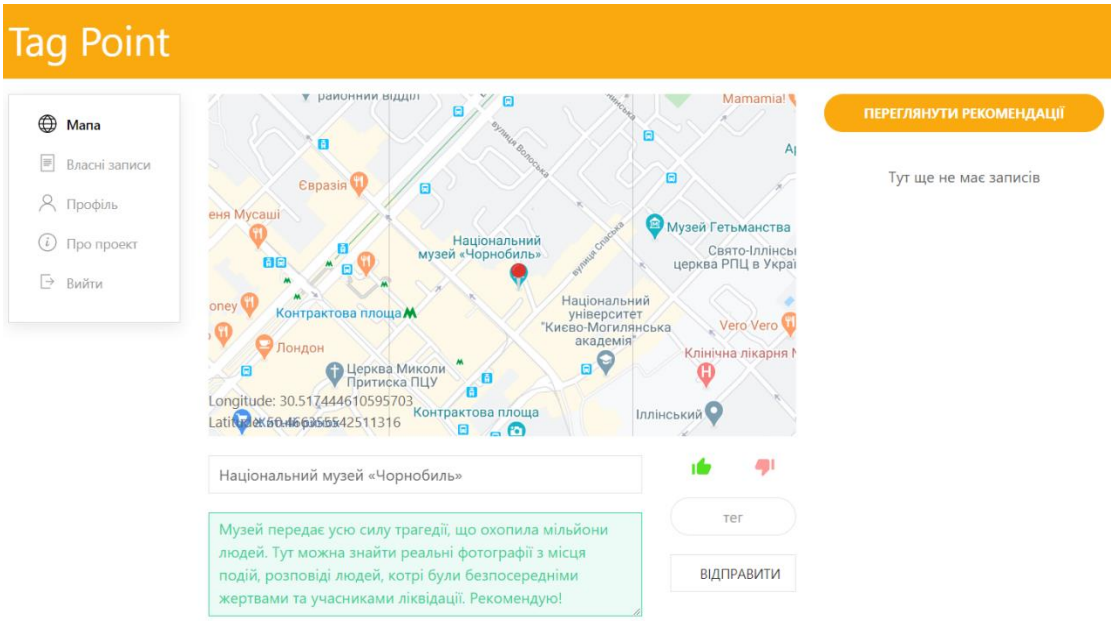


Рисунок 3.4 – Текстове повідомлення про місце

- додавання тегу до рекомендації відбувається у найменшому полі вводу. Потрібно вписати бажаний тег та натиснути «Пробіл», тег буде доданий до рекомендації та відображений під текстовим повідомленням;

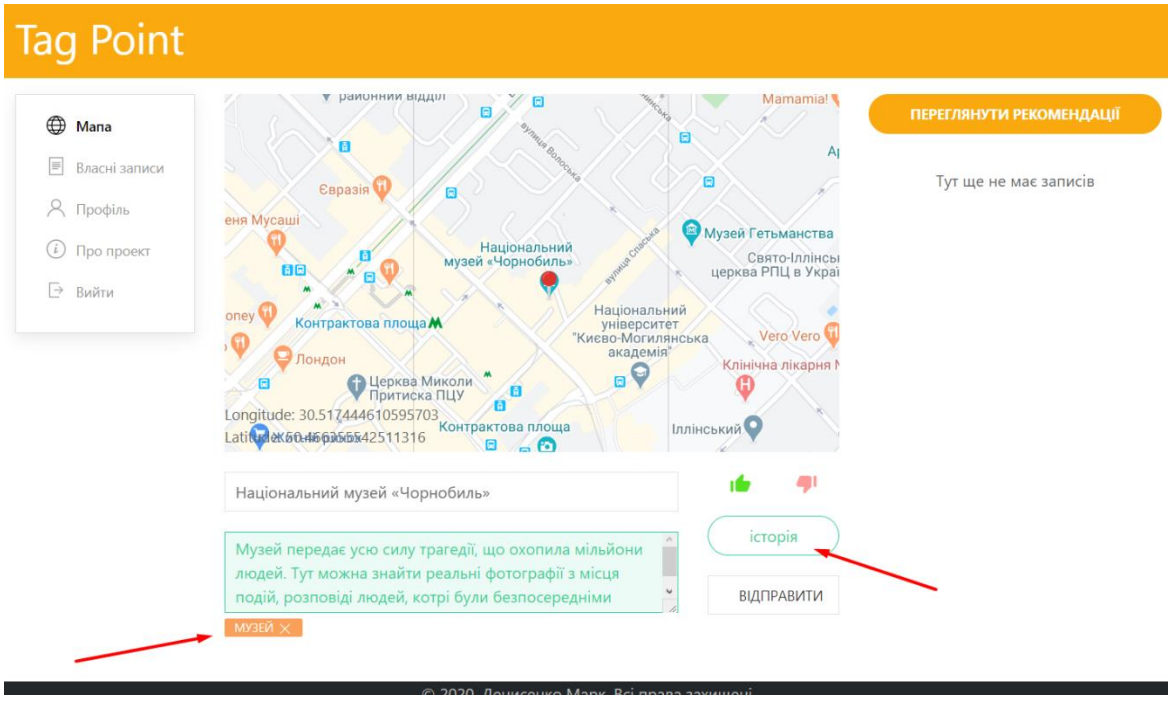


Рисунок 3.5 – Додавання тегу до рекомендації

- створення рекомендації відбувається при натисканні на кнопку «Відправити» та після обробки сервером користувачеві повернеться рекомендація, де можна буде побачити час створення та ім'я того, хто її створив.

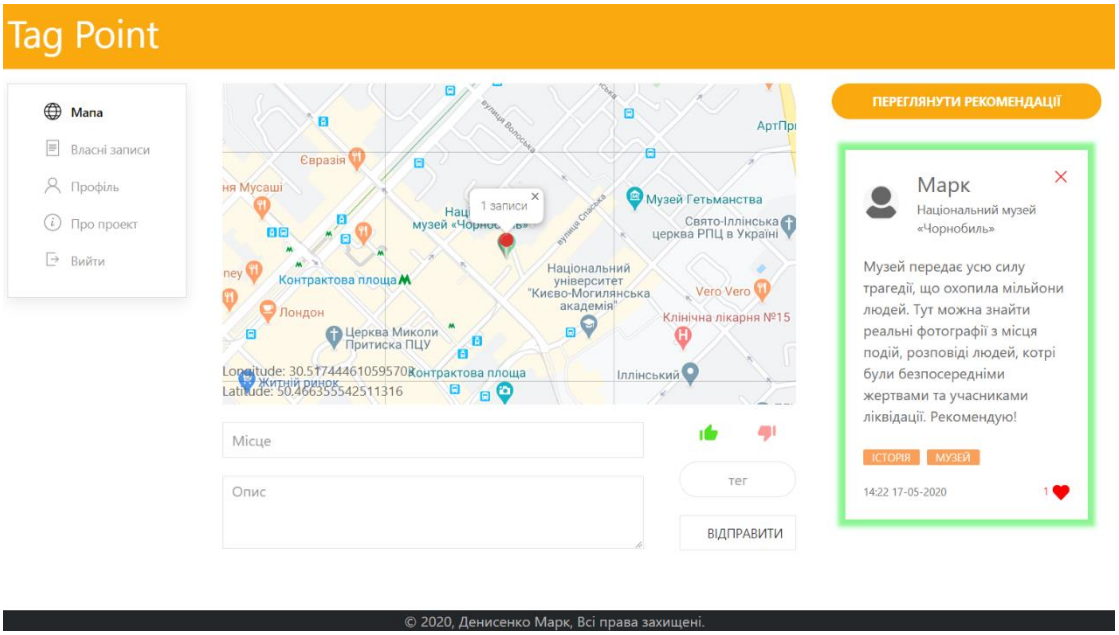


Рисунок 3.6 – Створення рекомендації

Рекомендація з’явилась та не зникає після оновлення сторінки та повторного пошуку. Її можна вподобати, а також видалити. Вона залишається закріпленою за створеною географічною точкою, та може бути знайдена будь-яким користувачем.

4 ВПРОВАДЖЕННЯ ТА СУПРОВІД ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Розгортання програмного забезпечення

Запуск геоінформаційного сервісу потребує ряд заздалегідь встановлених програмних компонентів. Для серверної частини необхідним є .NET Core Runtime та .NET Core SDK версії 3.1, який може бути розгорнутий на будь-якій платформі, та Visual Studio 2019 (за бажанням), що спростить процес розгортання. База даних потребує встановленого SQL Server або ж може бути використане хмарне рішення SQL Azure. Для успішного запуску клієнтської частини потрібно мати Node.js версії 12.16 та пакетного менеджера для завантаження необхідних бібліотек при побудові.

Коли усе необхідне програмне забезпечення встановлено, то наступним кроком у розгортанні є завантаження залежностей серверної частини за допомогою NuGet. Visual Studio зробить це в автоматичному режимі при відкритті або ж можна виконати команду у консолі «dotnet restore». Після завантаження усіх залежностей, потрібно змінити строку підключення до бази даних у файлі appsettings.json і серверне програмне забезпечення готове до розгортання та може бути запущено командою «dotnet run».

Клієнтська частина запускається за схожою схемою. Для початку необхідно завантажити усі залежності командою «npm install», по завершенню залишається запустити застосунок командою «npm run start».

4.2 Робота з програмним забезпеченням

Для початку роботи з сервісом у користувача має бути встановлений браузер та наявне з’єднання з Інтернетом. У наявному браузері потрібно ввести та відкрити посилання за яким розгорнуто програмне забезпечення. Після цього користувач може почати використовувати усі можливості інформаційної системи.

Детальну інформацію по роботі з програмним забезпеченням можна знайти в додатку Керівництво користувача».

ВИСНОВКИ

Підсумовуючи виконання даної дипломної роботи можна сказати, що у процесі створення програмного забезпечення було осягнуто етап аналізу предметної області, виділено множину необхідної функціональності для користувача, розглянуто наявні рішення та їх проблем, узагальнено існуючі недоліки та реалізовано шляхи їх вирішення.

Перед створенням сервісу надання рекомендацій проведено етап концептуального моделювання процесів роботи та детально описано архітектурні рішення підібрані відповідно до потреб розробки. Програмний код супроводжується діаграмами класів та взаємозв’язків між ними.

Для готового програмного забезпечення описано способи тестування у автоматизованому та ручному форматах. Також наведено кроки побудови та розгортання сервісу.

Розробка охопила проектування та створення бази даних, серверної та клієнтської частин, що укупі представляє єдину геоінформаційну систему з надання рекомендацій.

ПЕРЕЛІК ПОСИЛАНЬ

1) Evans E. Domain-Driven Design — Tackling Complexity in the Heart of Software / Eric Evans., 2004. – С.

2) Роберт С. М. Чистий код / С. М. Мартін – К.: Фабула, 2019. – 416 с.

3) Бочелюк В.Й. Дозвіллєзнавство: навч. посіб. / В.Й. Бочелюк, В.В. Бочелюк. – К.: Центр навч. літератури, 2006. – 208 с.

4) Інтеграція інтерактивної мапи [Електронний ресурс]: (Стаття) / Google Maps Platform – Електрон. дан. (1 файл). – 2019. – Режим доступу: https://developers.google.com/maps/documentation/javascript/tutorial. – Назва з екрана.

5) IEEE 829 - 1998 Standart for Software Test Documentation [Електронний ресурс]. – 2019. – Режим доступу до ресурсу: https://standards.ieee.org/standard/829-1998.html. – Назва з екрана.

6) How the Geolocation works [Електронний ресурс]: (Стаття) / Users Insights. – Електрон. Дан. (1 файл). – 2015. – Режим доступу: https://usersinsights.com/how-the-geolocation-works/. – Назва з екрана.

7) J. Bao, Y. Zheng, and M. F. Mokbel. Location-based and preference-aware recommendation using sparse geo-socialn etworking data, 2012 – 56 с.

8) J. J. Levandoski, M. Sarwat, A. Eldawy, and M. F. Mokbel. Lars: A location-aware recommender system., 2012 – 134 с.

9) Skiena S. The Algorithm Design Manual / Steven Skiena., 2008. – 748 с.

10) Angular Documentation [Електронний ресурс]: (Стаття) / – 2010 - 2020. – Режим доступу до ресурсу: https://angular.io/docs/. – Назва з екрана.

11) Material design [Електронний ресурс]: (Стаття) / – 2018. – Режим доступу до ресурсу: https://material.io/. – Назва з екрана.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ГЕОІНФОРМАЦІЙНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ НАДАННЯ РЕКОМЕНДАЦІЙ З ПОШУКУ МІСЦЬ ДОЗВІЛЛЯ

Технічне завдання

КПІ.ІП-6108.045490.03.81

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Є.А. Недашківський

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____ М. О. Денисенко

Київ – 2020 року

ЗМІСТ

1 НАЙМЕНУВАННЯ ТА ГАЛУЗЬ ЗАСТОСУВАННЯ

Назва розробки: Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля.

Галузь застосування:

Наведене технічне завдання поширюється на розробку геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля [КПІ.ІП-6108.045490.03.81], котре використовується для отримання користувачами рекомендацій стосовно місць дозвілля опираючись на географічне розташування та призначена для полегшення повсякденної задачі пошуку нових місць та закладів з метою провести вільний час.

2 ПІДСТАВА ДЛЯ РОЗРОБКИ

Підставою для розробки геоінформаційного програмного забезпечення для надання рекомендацій з пошуку місць дозвілля є завдання на дипломне проектування, затверджене кафедрою автоматизованих систем обробки інформації та управління Національного технічного університету України «Київський політехнічний інститут імені Ігоря Сікорського» («КПІ ім. Ігоря Сікорського»).

3 ПРИЗНАЧЕННЯ РОЗРОБКИ

Розробка призначена для надання персональних рекомендацій її користувачам стосовно місць дозвілля в залежності від географічного розташування.

Метою розробки є зменшення складності та часу необхідного для пошуку нових місць або закладів для проведення вільного часу.

4 ВИМОГИ ДО ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

4.1 Вимоги до функціональних характеристик

4.1.1 Програмне забезпечення повинно забезпечувати виконання наступних основних функції:

4.1.1.1 Для користувача:

– авторизація;
– реєстрація;
– редагування інформації власного профілю;
– створення рекомендацій;
– видалення власних;
– перегляд створених рекомендацій;
– уподобання рекомендацій;
– перегляд персональних рекомендацій;
– пошук рекомендацій за тегами;
– перегляд статистики власних рекомендацій.

4.1.2 Розробка виконати на платформі Windows

4.1.3 Додаткові вимоги:

– відображати географічну мапу для поліпшення можливості графічного пошуку;
– сортування власних рекомендацій за певними характеристиками;
– пошук власних рекомендацій за ключовим словом.

4.2 Вимоги до надійності

4.2.1 Передбачити контроль введення інформації

З метою уникнення помилок при взаємодії з програмним забезпеченням на наявність значення мусять бути перевіреними: поля логіну та паролю при автентифікації, поля імені, емейлу, логіну та паролю при реєстрації, текстового повідомлення рекомендації на головній сторінці при її створенні.

4.2.2 Передбачити захист від некоректних дій користувача

Щоб убезпечити користувача від випадкового видалення рекомендацій при спробі видалити її виводиться повідомлення для підтвердження дії.

4.2.3 Передбачити уникнення втрат персональних даних користувачів

Комплексна взаємодія зміни даних відбувається з використанням транзакцій, щоб уникнути неузгодженості даних. Проблема надлишковості даних вирішується приведенням бази даних до третьої нормальної форми.

4.2.4 Забезпечити цілісність інформації в базі даних

4.3 Умови експлуатації

4.3.1 Умови експлуатації згідно СанПін 2.2.2.542 – 96

Не висуваються

4.3.2 Обслуговування

Програмне забезпечення повністю автоматизоване та не вимагає ніякого обслуговування.

4.4 Вимоги до складу і параметрів технічних засобів

4.4.1 Програмне забезпечення повинно функціонувати на IBM-сумісних персональних комп‘ютерах

4.4.2 Мінімальна конфігурація технічних засобів

4.4.2.1 Тип процесор

64-розрядний процесор з тактовою частотою не нижче 1.4 ГГц

4.4.2.2 Об‘єм ОЗП

Не менше 2 ГБ

4.4.2.3 Об’єм жорсткого диску

Не менше 1 ГБ.

4.5 Вимоги до інформаційної та програмної сумісності

а) Програмне забезпечення повинно працювати під управлінням операційних систем сімейства Windows 10.

б) Вхідні дані для основної частині програмного забезпечення мусять бути у форматі HTTP запитів з URL параметрами або ж у JSON форматі тілі запиту, що надсилаються з клієнтської частини.

Клієнтська частина у свою чергу має графічний інтерфейс користувача для взаємодії з системою. Користувач повинен лише заповнювати необхідні поля форм або графічно обирати потрібно місцевість для пошуку.

в) Вихідним результатом основного програмного забезпечення є відповіді на HTTP-запити у форматі JSON. Клієнтська частина графічно відображає користувачеві отримані дані з серверу.

г) Серверна частина побудована на мові C# та платформі .NET Core, клієнтський застосунок створений за допомогою Angular (JavaScript фреймворк) та частина бази даних підтримується MS SQL Server.

4.6 Вимоги до маркування та пакування

Вимоги до маркування та пакування не висуваються.

4.7 Вимоги до транспортування та зберігання

Вимоги до транспортування та зберігання не висуваються.

4.8 Спеціальні вимоги

Має бути створена установча версію програмного забезпечення.

5 ВИМОГИ ДО ПРОГРАМНОЇ ДОКУМЕНТАЦІЇ

5.1 Попередній склад програмної документації

а) Супроводжувальна документація

1) Пояснювальна записка.

2) Технічне завдання.

3) Керівництво користувача.

4) Програма та методика тестування.

б) Довідникова документація

1) Програмні модулі, котрі розробляються, повинні бути задокументовані, тобто тексти програм повинні містити всі необхідні коментарі.

2) Програмне забезпечення повинно мати довідникову систем.

в) Графічна документація

1) Схема структурна бази даних.

2) Креслення екранних форм.

3) Схема структурна варіантів використання.

6 СТАДІЇ І ЕТАПИ РОЗРОБКИ

№	Назва етапу	Строк,	Звітність
1.	Вивчення літератури за тематикою проекту	*06.04.2020*	
2.	Розробка технічного завдання	*10.04.2020*	Технічне завдання
3.	Аналіз вимог та уточнення специфікацій	*12.04.2020*	Специфікації програмного забезпечення
4.	Проектування структури програмного забезпечення, проектування компонентів	*13.04.2020*	Схема структурна програмного забезпечення та специфікація компонентів
5.	Програмна реалізація програмного забезпечення	*16.04.2020*	Тексти програмного забезпечення
6.	Тестування програмного забезпечення	*22.04.2020*	Тести, результати тестування
7.	Розробка матеріалів текстової частини проекту	*29.04.2020*	Пояснювальна записка.
8.	Розробка матеріалів графічної частини проекту	*21.05.2020*	Графічний матеріал проекту
9.	Оформлення технічної документації проекту	*25.05.2020*	Технічна документація

7 ПОРЯДОК КОНТРОЛЮ ТА ПРИЙМАННЯ

7.1 Види випробувань

Тестування розробленого програмного продукту виконується відповідно до “Програми та методики тестування”.

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ГЕОІНФОРМАЦІЙНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ НАДАННЯ РЕКОМЕНДАЦІЙ З ПОШУКУ МІСЦЬ ДОЗВІЛЛЯ

Програма та методики тестування

КПІ.ІП-6108.045490.04.51

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Є. А. Недашківський

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____М. О. Денисенко

Київ – 2020 року

ЗМІСТ

1 ОБ'ЄКТ ВИПРОБУВАНЬ

Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля яке працює у режимі клієнт-сервер, де клієнт працює на JavaScript під управлінням Angular, а сервер керується кодом написаним на C# платформи ASP.NET Core.

2 МЕТА ТЕСТУВАННЯ

Тестування проводиться з метою оцінки якості програмного забезпечення та відповідності існуючого функціоналу до вимог. До перевірки під час тестування входять такі частини:

- функціональна логіка дії сторінок клієнтської програми;
- можливість реєстрації та автентифікації;
- перевірка захищеного з’єднання між клієнтом та сервером;
- перегляд персональних рекомендацій;
- перегляд та редагування профілю користувача;
- створення та взаємодія з існуючими рекомендаціями.

3 МЕТОДИ ТЕСТУВАННЯ

Для тестування програмного забезпечення використовуються:

- ручне тестування за допомогою графічного інтерфейсу;
- автоматизоване тестування за допомогою юніт-тестів.

4 ЗАСОБИ ТА ПОРЯДОК ТЕСТУВАННЯ

Тестування програмного забезпечення відбувається шляхом:

– перевірка відповідності функціональних вимог до роботи інформаційної системи;

– запуску коду тестування у вигляді юніт-тестів;

– випробування інформаційної системи з декількома web-браузерами;

– перевірка недопустимих та граничних значень за допомогою ручного вводу;

– тестування графічного інтерфейсу.

Факультет інформатики та обчислювальної техніки
Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”
В.о. завідувача кафедри
_____ Олександр ПАВЛОВ
“___” _____ 2020 р.

ГЕОІНФОРМАЦІЙНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ НАДАННЯ РЕКОМЕНДАЦІЙ З ПОШУКУ МІСЦЬ ДОЗВІЛЛЯ

Опис програми

КПІ.ІП-6108.045490.05.13

“ПОГОДЖЕНО”
Керівник проєкту:
_____ Є. А. Недашківський

Нормоконтроль:
_____ К.І. Ліщук

Виконавець:
_____М. О. Денисенко

Київ – 2020 року

Тексти програмного коду

Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля

(Найменування програми (документа))

DVD-R

(Вид носія даних)

29 арк, 124 Кб

(Обсяг програми (документа) , арк.,)

Київ – 2020

```
using Database.Country;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Database;
using Model.Models;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Application.Geolocation
{
   public class GeolocationApplicationService : IGeolocationApplicationService
   {
      //private readonly IUnitOfWork _unitOfWork;
      private readonly ICountryRepository _countryRepository;

      public GeolocationApplicationService(
         //IUnitOfWork unitOfWork,
         ICountryRepository userRepository)
      {
         //_unitOfWork = unitOfWork;
         _countryRepository = userRepository;
      }

      public async Task<IDataResult<IEnumerable<CountryModel>>> GetCountriesAsync()
      {
         var countries = await _countryRepository.ListAsync();
         return DataResult<IEnumerable<CountryModel>>.Success(countries
            .OrderBy(c => c.Country)
            .Select(c => new CountryModel
            {
               Id = c.Id,
               Country = c.Country,
               CountryCode = c.CountryCode
            }));
      }
   }
}
using Database.Point;
using Database.Post;
using Database.Vote;
using Domain.Point;
using Domain.Post;
using Domain.ValueObjects;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Database;
using DotNetCoreArchitecture.Domain;
using Microsoft.EntityFrameworkCore;
using Model.Models.Map;
using Model.Models.Post;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
```

```
namespace Application.Post
{
   public class PostApplicationService : IPostApplicationService
   {
      private readonly IUnitOfWork _unitOfWork;
      private readonly IPostRepository _postRepository;
      private readonly IPointRepository _pointRepository;
      private readonly IUserRepository _userRepository;
      private readonly ILikeRepository _likesRepository;
      private readonly ITagRepository _tagRepository;
      private readonly IPostTagRepository _postTagRepository;

      public PostApplicationService(
         IUnitOfWork unitOfWork,
         IPostRepository postRepository,
         IPointRepository pointRepository,
         IUserRepository userRepository,
         ILikeRepository likesRepository,
         ITagRepository tagRepository,
         IPostTagRepository postTagRepository)
      {
         _unitOfWork = unitOfWork;
         _postRepository = postRepository;
         _pointRepository = pointRepository;
         _userRepository = userRepository;
         _likesRepository = likesRepository;
         _tagRepository = tagRepository;
         _postTagRepository = postTagRepository;
      }

      public async Task<IDataResult<long>> CreatePostAsync(PostModel post)
      {
         var user = _userRepository.FirstOrDefault(u => u.Id == post.UserId);

         PointEntity point = await _pointRepository.FirstOrDefaultWhereIncludeAsync(p => p.Id ==
post.Marker.Id, p => p.Posts);
         if (point is null)
         {
            point = new PointEntity
            {
               Coordinate = new Coordinate
               {
                  Latitude = post.Marker.Latitude,
                  Longitude = post.Marker.Longitude
               }
            };
            await _pointRepository.AddAsync(point);
            point.User = user;
         }

         var newPost = new PostEntity
         {
            Message = post.Message,
            Recommended = post.Recommended,
            PostDate = DateTime.UtcNow,
```

```
            Location = post.Location
        };

        await _postRepository.AddAsync(newPost);
        newPost.Point = point;
        newPost.User = user;

        _tagRepository.CreateTags(post.Tags, user.Id);

        await _unitOfWork.SaveChangesAsync();

        var postTags = _tagRepository.GetTagsByNames(post.Tags);
        await _postTagRepository.AddRangeAsync(postTags.Select(t => new PostTagEntity
        {
            PostId = newPost.Id,
            TagId = t.Id
        }));

        await _unitOfWork.SaveChangesAsync();

        return DataResult<long>.Success(newPost.Id);
    }

    public async Task<IResult> DeletePostAsync(long id)
    {
        var post = await _postRepository.FirstOrDefaultWhereIncludeAsync(p => p.Id == id, post =>
post.Point);
        await _postRepository.DeleteAsync(id);
        await _unitOfWork.SaveChangesAsync();

        var point = await _pointRepository.FirstOrDefaultWhereIncludeAsync(p => p.Id == post.Point.Id, p
=> p.Posts);
        if (!point.Posts.Any())
        {
            await _pointRepository.DeleteAsync(post.Point.Id);
        }

        await _unitOfWork.SaveChangesAsync();

        return Result.Success();
    }

    public async Task<IDataResult<PostModel>> GetPostAsync(long id)
    {
        var post = await _postRepository.SingleOrDefaultWhereIncludeAsync(p => p.Id == id, p => p.User);

        return DataResult<PostModel>.Success(CreatePostModel(post));
    }

    public async Task<IDataResult<IEnumerable<MarkerModel>>>
GetMarkerWithPostsNearAsync(Coordinate center, double radius, long userId)
    {
        var markersInRadius = await _pointRepository.GetPointsInRadius(center, radius);

        await _pointRepository.IncrementPointViews(markersInRadius.Select(p => p.Id));
        await _unitOfWork.SaveChangesAsync();
```

```
            var postsIds = markersInRadius.SelectMany(m => m.Posts.Select(p => p.Id)).ToList();
            var posts = await _postRepository.Posts
               .AsNoTracking()
               .Include(p => p.Likes)
               .Include(p => p.Point)
               .Include(p => p.Tags)
               .Include(p => p.User)
               .ThenInclude(u => u.Avatar)
               .Where(p => postsIds.Contains(p.Id))
               .ToListAsync();

            var existedTags = await _tagRepository.ListAsync();
            return DataResult<IEnumerable<MarkerModel>>.Success(markersInRadius.Select(m =>
               new MarkerModel
               {
                  Id = m.Id,
                  Latitude = m.Coordinate.Latitude,
                  Longitude = m.Coordinate.Longitude,
                  Posts = posts.Where(p => p.Point.Id == m.Id)
                     .Select(p =>
                     {
                        var returnPost = CreatePostModel(p);
                        returnPost.Liked = p.Likes.Any(like => like.UserId == userId);
                        returnPost.TimesLiked = p.Likes.Count;
                        returnPost.Editable = p.User.Id == userId;
                        returnPost.Tags = existedTags.Where(t => p.Tags.Any(pt => pt.TagId == t.Id)).Select(t =>
t.Tag);

                        return returnPost;
                     })
               }));
         }

         public async Task<IDataResult<IEnumerable<PostModel>>> GetUserPosts(long userId, PostsRequest
postsRequest)
         {
            var posts = await _postRepository.ListWhereIncludeAsync(post => post.User.Id == userId, p =>
p.User, p => p.User, p => p.Likes, p => p.Point);

            var postsResult = posts.Select(p =>
            {
               var post = CreatePostModel(p);
               //post.Views = p.Point.TotalViews;

               return post;
            });

            if (postsRequest.OrderByDateDesc.HasValue)
            {
               postsResult = postsRequest.OrderByDateDesc.Value
                  ? postsResult.OrderByDescending(p => p.CreationDate)
                  : postsResult.OrderBy(p => p.CreationDate);
            }

            if (postsRequest.OrderByLikesDesc.HasValue)
```

```
            {
                postsResult = postsRequest.OrderByLikesDesc.Value
                    ? postsResult.OrderByDescending(p => p.TimesLiked)
                    : postsResult.OrderBy(p => p.TimesLiked);
            }

            if (!string.IsNullOrWhiteSpace(postsRequest.Keyword))
            {
                postsResult = postsResult.Where(p => p.Message.Contains(postsRequest.Keyword)
                    || p.Location.Contains(postsRequest.Keyword)
                    || p.TimesLiked.ToString().Contains(postsRequest.Keyword));
            }

            return DataResult<IEnumerable<PostModel>>.Success(postsResult);
        }

        public async Task<int> ToggleLikePostAync(long postId, long userId)
        {
            var user = await _userRepository.SingleOrDefaultWhereIncludeAsync(u => u.Id == userId, u =>
u.Likes);
            var post = await _postRepository.SingleOrDefaultWhereIncludeAsync(p => p.Id == postId, p =>
p.Likes);

            var like = post.Likes.FirstOrDefault(like => like.UserId == userId);
            if (like is null)
            {
                like = new Domain.Vote.LikeEntity
                {
                    PostId = postId,
                    UserId = userId,
                };
                await _likesRepository.AddAsync(like);
                post.Likes.Add(like);
                user.Likes.Add(like);
            }
            else
            {
                _likesRepository.Remove(postId, userId);
            }

            await _unitOfWork.SaveChangesAsync();
            post = await _postRepository.SingleOrDefaultWhereIncludeAsync(p => p.Id == postId, p =>
p.Likes);

            return post.Likes.Count;
        }

        private PostModel CreatePostModel(PostEntity post)
        {
            return new PostModel
            {
                Id = post.Id,
                Message = post.Message,
                Location = post.Location,
                Recommended = post.Recommended,
                CreationDate = post.PostDate.ToUniversalTime(),
```

```
                TimesLiked = post.Likes.Count,
                UserId = post.User.Id,
                Username = post.User.Username,
                UserAvatar = post.User.Avatar?.Avatar,
                Views = post.TotalViews,
            };
        }
    }
}
using Database.Country;
using Database.User;
using Domain.Country;
using Domain.User;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Database;
using DotNetCoreArchitecture.Domain;
using DotNetCoreArchitecture.Infra;
using DotNetCoreArchitecture.Model;
using Model.Models.User;
using System.Collections.Generic;
using System.Threading.Tasks;

namespace DotNetCoreArchitecture.Application
{
    public sealed class UserApplicationService : IUserApplicationService
    {
        private readonly ISignInService _signInService;
        private readonly IUnitOfWork _unitOfWork;
        private readonly IUserRepository _userRepository;
        private readonly ICountryRepository _countryRepository;
        private readonly IUserAvatarRepository _userAvatarRepository;

        public UserApplicationService
        (
            ISignInService signInService,
            IUnitOfWork unitOfWork,
            IUserRepository userRepository,
            ICountryRepository countryRepository,
            IUserAvatarRepository userAvatarRepository
        )
        {
            _signInService = signInService;
            _unitOfWork = unitOfWork;
            _userRepository = userRepository;
            _countryRepository = countryRepository;
            _userAvatarRepository = userAvatarRepository;
        }

        public async Task<IDataResult<long>> AddAsync(AddUserModel addUserModel)
        {
            var validation = new AddUserModelValidator().Validate(addUserModel);

            if (validation.Failed)
            {
                return DataResult<long>.Fail(validation.Message);
            }
```

```
            addUserModel.SignIn = _signInService.CreateSignIn(addUserModel.SignIn);

            var userEntity = UserFactory.Create(addUserModel);
            userEntity.Add();
            await _userRepository.AddAsync(userEntity);

            var country = _countryRepository.FirstOrDefault(country => country.Id ==
addUserModel.CountryId);
            userEntity.Country = country;

            await _unitOfWork.SaveChangesAsync();

            return DataResult<long>.Success(userEntity.Id);
        }

        public async Task<IResult> SetAvatarAsync(long userId, BinaryFile avatar)
        {
            var user = await _userRepository.FirstOrDefaultAsync(u => u.Id == userId);
            var newAvatar = new UserAvatarEntity
            {
                Filename = avatar.Name,
                Avatar = avatar.Bytes
            };

            await _userAvatarRepository.DeleteAsync(avatar => avatar.UserId == userId);
            user.Avatar = newAvatar;
            await _userRepository.UpdateAsync(user.Id, user);

            await _unitOfWork.SaveChangesAsync();

            return Result.Success();
        }

        public async Task<IResult> DeleteAsync(long id)
        {
            await _userRepository.DeleteAsync(id);

            await _unitOfWork.SaveChangesAsync();

            return Result.Success();
        }

        public async Task<IDataResult<TokenModel>> SignInAsync(SignInModel signInModel)
        {
            var validation = new SignInModelValidator().Validate(signInModel);

            if (validation.Failed)
            {
                return DataResult<TokenModel>.Fail(validation.Message);
            }

            var signedInModel = await _userRepository.SignInAsync(signInModel);

            validation = _signInService.Validate(signedInModel, signInModel);
```

```
            if (validation.Failed)
            {
                return DataResult<TokenModel>.Fail(validation.Message);
            }

            //var userLogModel = new UserLogModel(signedInModel.Id, LogType.SignIn);

            var tokenModel = _signInService.CreateToken(signedInModel);

            return DataResult<TokenModel>.Success(tokenModel);
        }

        public Task SignOutAsync(SignOutModel signOutModel)
        {
            //var userLogModel = new UserLogModel(signOutModel.Id, LogType.SignOut);
            return Task.CompletedTask;
        }

        public async Task<IResult> UpdateAsync(UpdateUserModel updateUserModel)
        {
            var validation = new UpdateUserModelValidator().Validate(updateUserModel);

            if (validation.Failed)
            {
                return Result.Fail(validation.Message);
            }

            var userEntity = await _userRepository.SelectAsync(updateUserModel.Id);

            if (userEntity == default)
            {
                return Result.Success();
            }

            userEntity.ChangeUsername(updateUserModel.Username);
            userEntity.ChangeEmail(updateUserModel.Email);
            userEntity.ChangePhone(updateUserModel.Phone);
            userEntity.ChangeGender(updateUserModel.Gender);
            userEntity.ChangeAbout(updateUserModel.About);
            userEntity.Country = _countryRepository.FirstOrDefault(country => country.Id ==
updateUserModel.CountryId);

            await _userRepository.UpdateAsync(userEntity.Id, userEntity);
            await _unitOfWork.SaveChangesAsync();

            return Result.Success();
        }

        public async Task InactivateAsync(long id)
        {
            var userEntity = UserFactory.Create(id);

            userEntity.Inactivate();

            await _userRepository.UpdateStatusAsync(userEntity);
```

```
    await _unitOfWork.SaveChangesAsync();
}

public async Task<PagedList<UserModel>> ListAsync(PagedListParameters parameters)
{
    return await _userRepository.ListAsync<UserModel>(parameters);
}

public async Task<IEnumerable<UserModel>> ListAsync()
{
    return await _userRepository.ListAsync<UserModel>();
}

public async Task<UserModel> SelectAsync(long id)
{
    return await _userRepository.SelectAsync<UserModel>(id);
}

public async Task<IDataResult<ProfileModel>> GetAsync(long id)
{
    var user = await _userRepository.FirstOrDefaultWhereIncludeAsync(u => u.Id == id,
        u => u.Country,
        u => u.Avatar,
        user => user.Posts);

    return DataResult<ProfileModel>.Success(new ProfileModel
    {
      Id = user.Id,
      Username = user.Username,
      Email = user.Email.Address,
      Phone = user.PhoneNumber,
      About = user.About,
      Gender = user.Gender,
      Country = user.Country?.Country,
      Avatar = user.Avatar?.Avatar,
      CountryId = user.Country?.Id ?? 0,
      RegistrationDate = user.RegisterDate,
      PostsNumber = user.Posts.Count
    });
}

public async Task<IResult> UpdatePasswordAsync(UserChangePassword updateUserModel)
{
    var user = await _userRepository.SelectAsync(updateUserModel.UserId);
    var signedInModel = await _userRepository.SignInAsync(new SignInModel
    {
       Login = user.SignIn.Login
    });

    var validation = _signInService.Validate(signedInModel, new SignInModel
    {
       Login = user.SignIn.Login,
       Password = updateUserModel.OldPassword
    });

    if (validation.Failed)
```

```
            {
                return Result.Fail("Старий пароль введено не вірно!");
            }

            var newSignInModel = _signInService.CreateSignIn(new SignInModel
            {
                Login = user.SignIn.Login,
                Password = updateUserModel.NewPassword
            });
            user.ChangeSignIn(new SignIn
                (
                    newSignInModel.Login,
                    newSignInModel.Password,
                    newSignInModel.Salt
                ));

            await _userRepository.UpdateAsync(user.Id, user);

            await _unitOfWork.SaveChangesAsync();

            return Result.Success();
        }
    }
}

using Domain.Post;
using Domain.Country;
using Domain.Point;
using Domain.User;
using Domain.Vote;
using DotNetCore.EntityFrameworkCore;
using Microsoft.EntityFrameworkCore;
using DotNetCoreArchitecture.Domain;

namespace DotNetCoreArchitecture.Database
{
    public sealed class Context : DbContext
    {
        public DbSet<PostEntity> Posts { get; set; }
        public DbSet<UserEntity> Users { get; set; }
        public DbSet<LikeEntity> Likes { get; set; }
        public DbSet<CountryEntity> Countries { get; set; }
        public DbSet<PointEntity> Points { get; set; }
        public DbSet<TagSelectionEntity> TagSelections { get; set; }

        public Context(DbContextOptions options) : base(options)
        {
            //Database.EnsureDeleted();
            Database.EnsureCreated();
            //ChangeTracker.QueryTrackingBehavior = QueryTrackingBehavior.NoTracking;
            //Database.Migrate();
        }

        protected override void OnModelCreating(ModelBuilder builder)
        {
            builder.ApplyConfigurationsFromAssembly();
```

```
            builder.Seed();
        }
    }
}
using System.Threading.Tasks;

namespace DotNetCoreArchitecture.Database
{
    public sealed class UnitOfWork : IUnitOfWork
    {
        private readonly Context _context;

        public UnitOfWork(Context context)
        {
            _context = context;
        }

        public Task<int> SaveChangesAsync()
        {
            return _context.SaveChangesAsync();
        }
    }
}
using Domain.Point;
using Domain.ValueObjects;
using DotNetCore.Repositories;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace Database.Point
{
    public interface IPointRepository : IRelationalRepository<PointEntity>
    {
        Task<IEnumerable<PointEntity>> GetPointsInRadius(Coordinate center, double radius);
        Task IncrementPointViews(IEnumerable<long> pointIds);
    }
}
using Domain.Post;
using DotNetCore.Repositories;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Database.Post
{
    public interface IPostRepository : IRelationalRepository<PostEntity>
    {
        public IQueryable<PostEntity> Posts { get; }
    }
}

using Domain.Post;
using DotNetCore.Repositories;
```

```
using System;
using System.Collections.Generic;
using System.Text;

namespace Database.Post
{
   public interface ITagRepository : IRelationalRepository<TagEntity>
   {
      void CreateTags(IEnumerable<string> tags, long userId);
      IEnumerable<TagEntity> GetTagsByNames(IEnumerable<string> tags);
      IEnumerable<TagEntity> GetTags(string keyword);
   }
}
using Domain.Post;
using DotNetCore.Repositories;
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading.Tasks;

namespace Database.Post
{
   public interface ITagSelectionRepository : IRelationalRepository<TagSelectionEntity>
   {
      Task CreateTagSelectionForUser(string tag, long userId);
      Task CreateTagSelectionForUser(long tagId, long userId);
      Task<IEnumerable<string>> GetUserTagSelection(long id);
   }
}
using Domain.Post;
using DotNetCore.EntityFrameworkCore;
using DotNetCoreArchitecture.Database;
using Microsoft.EntityFrameworkCore;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;

namespace Database.Post
{
   public class TagRepository : EntityFrameworkCoreRelationalRepository<TagEntity>, ITagRepository
   {
      public TagRepository(Context context) : base(context)
      {
      }

      public void CreateTags(IEnumerable<string> tags, long userId)
      {
         var tagsToCreate = tags
            .Where(t => t != null && !string.IsNullOrWhiteSpace(t))
            .Select(t => new TagEntity
            {
               Tag = t.Trim().ToLowerInvariant(),
               CreatedById = userId
            })
            .ToList();
```

```
            var existedTags = ListInclude().Select(t => t.Tag);
            var needCreate = tagsToCreate.Where(t => !existedTags.Contains(t.Tag));

            AddRange(needCreate);
        }

        public IEnumerable<TagEntity> GetTags(string keyword)
        {
            var formattedKeyword = keyword.Trim().ToLowerInvariant();
            return ListWhereInclude(t => t.Tag.ToLowerInvariant().Contains(formattedKeyword));
        }

        public IEnumerable<TagEntity> GetTagsByNames(IEnumerable<string> tags)
        {
            var formattedTags = tags.Select(t => t.Trim().ToLowerInvariant()).ToList();
            return ListWhereInclude(t => formattedTags.Contains(t.Tag));
        }
    }
}
using Domain.Post;
using DotNetCore.EntityFrameworkCore;
using DotNetCoreArchitecture.Database;
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using Microsoft.EntityFrameworkCore;

namespace Database.Post
{
    public class TagSelectionRepository : EntityFrameworkCoreRelationalRepository<TagSelectionEntity>,
ITagSelectionRepository
    {
        private readonly ITagRepository tagRepository;
        private readonly Context context;

        public TagSelectionRepository(ITagRepository tagRepository, Context context) : base(context)
        {
            this.tagRepository = tagRepository;
            this.context = context;
        }

        public async Task CreateTagSelectionForUser(long tagId, long userId)
        {
            await AddAsync(new TagSelectionEntity
            {
                TagId = tagId,
                UserId = userId
            });
            await context.SaveChangesAsync();
        }

        public async Task CreateTagSelectionForUser(string tagName, long userId)
        {
            var tag = tagRepository.GetTagsByNames(new List<string> { tagName }).FirstOrDefault()?.Id;
```

```
            if (!tag.HasValue)
            {
                tagRepository.CreateTags(new List<string> { tagName }, userId);
                await context.SaveChangesAsync();

                tag = tagRepository.GetTagsByNames(new List<string> { tagName }).FirstOrDefault()?.Id;
            }

            await CreateTagSelectionForUser(tag.Value, userId);
        }

        public async Task<IEnumerable<string>> GetUserTagSelection(long id)
        {
            return await context.TagSelections
                .Where(t => t.UserId == id)
                .GroupBy(t => t.Tag.Tag)
                .OrderByDescending(t => t.Count())
                .Select(t => t.Key)
                .ToListAsync();
        }
    }
}
using DotNetCore.Repositories;
using DotNetCoreArchitecture.Domain;
using DotNetCoreArchitecture.Model;
using System.Threading.Tasks;

namespace DotNetCoreArchitecture.Database
{
    public interface IUserRepository : IRelationalRepository<UserEntity>
    {
        Task<SignedInModel> SignInAsync(SignInModel signInModel);

        Task UpdateStatusAsync(UserEntity userEntity);
    }
}
using DotNetCore.EntityFrameworkCore;
using DotNetCoreArchitecture.Domain;
using DotNetCoreArchitecture.Model;
using System.Threading.Tasks;

namespace DotNetCoreArchitecture.Database
{
    public sealed class UserRepository : EntityFrameworkCoreRelationalRepository<UserEntity>,
IUserRepository
    {
        public UserRepository(Context context) : base(context)
        {
        }

        public Task<SignedInModel> SignInAsync(SignInModel signInModel)
        {
            return SingleOrDefaultAsync<SignedInModel>
            (
                userEntity =>
```

```
                userEntity.SignIn.Login == signInModel.Login &&
                userEntity.Status == Status.Active
            );
        }

        public Task UpdateStatusAsync(UserEntity userEntity)
        {
            return UpdatePartialAsync(userEntity.Id, new { userEntity.Status });
        }
    }
}
using DotNetCoreArchitecture.Domain;
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.Country
{
    public class CountryEntity
    {
        public int Id { get; set; }
        public string Country { get; set; }
        public string CountryCode { get; set; }

        public ICollection<UserEntity> Users { get; private set; }

        public void AddUser(UserEntity newUser)
        {
            Users.Add(newUser);
        }
    }
}
using Domain.Post;
using Domain.ValueObjects;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Domain;
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.Point
{
    public class PointEntity : Entity
    {
        public long TotalViews { get; set; }

        public Coordinate Coordinate { get; set; }

        public UserEntity User { get; set; }
        public long UserId { get; set; }
        public DateTime Created { get; set; } = DateTime.Now;

        public ICollection<PostEntity> Posts { get; private set; }

        public void AddPost(PostEntity newPost)
        {
```

```
            Posts.Add(newPost);
        }
    }
}
using Domain.Point;
using Domain.Vote;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Domain;
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.Post
{
    public class PostEntity : Entity
    {
        public string Message { get; set; }
        public bool Recommended { get; set; }
        public string Location { get; set; }
        public DateTime PostDate { get; set; }
        public UserEntity User { get; set; }
        public PointEntity Point { get; set; }
        public long TotalViews { get; set; }
        public ICollection<LikeEntity> Likes { get; set; } = new List<LikeEntity>();
        public ICollection<PostTagEntity> Tags { get; set; } = new List<PostTagEntity>();
    }
}
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.Post
{
    public class PostTagEntity
    {
        public PostEntity Post { get; set; }
        public long PostId { get; set; }

        public TagEntity Tag { get; set; }
        public long TagId { get; set; }
    }
}
using DotNetCore.Objects;
using DotNetCoreArchitecture.Domain;
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.Post
{
    public class TagEntity : Entity
    {
        public string Tag { get; set; }
        public DateTime Created { get; set; } = DateTime.Now;
        public UserEntity CreatedBy { get; set; }
        public long CreatedById { get; set; }
```

```
        public ICollection<PostTagEntity> Posts { get; set; }
        public ICollection<TagSelectionEntity> TagSelections { get; set; }
    }
}
using DotNetCore.Objects;
using DotNetCoreArchitecture.Domain;
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.Post
{
    public class TagSelectionEntity : Entity
    {
        public TagEntity Tag { get; set; }
        public long TagId { get; set; }
        public DateTime Created { get; set; }
        public UserEntity User { get; set; }
        public long UserId { get; set; }
    }
}
using DotNetCore.Objects;
using DotNetCoreArchitecture.Domain;
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.User
{
    public class UserAvatarEntity : Entity
    {
        public string Filename { get; set; }
        public byte[] Avatar { get; set; }
        public UserEntity User { get; set; }
        public long UserId { get; set; }
        public DateTime UploadedTime { get; set; }
    }
}
using Domain.Country;
using Domain.Point;
using Domain.Post;
using Domain.User;
using Domain.Vote;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Model;
using Model;
using System;
using System.Collections.Generic;

namespace DotNetCoreArchitecture.Domain
{
    public class UserEntity : Entity
    {
        public UserEntity
        (
```

```
    long id,
    string username,
    Email email,
    SignIn signIn,
    Roles roles,
    Status status,
    Gender gender,
    string phoneNumber,
    string about = null
)
{
    Id = id;
    Username = username;
    Email = email;
    SignIn = signIn;
    Roles = roles;
    Status = status;
    Gender = gender;
    About = about;
    PhoneNumber = phoneNumber;
}

public UserEntity(long id)
{
    Id = id;
}

public string Username { get; private set; }
public Gender Gender { get; private set; }
public Email Email { get; private set; }
public SignIn SignIn { get; private set; }
public Roles Roles { get; private set; }
public Status Status { get; private set; }
public string About { get; private set; }
public string PhoneNumber { get; private set; }
public CountryEntity Country { get; set; }
public UserAvatarEntity Avatar { get; set; }
public DateTime RegisterDate { get; set; } = DateTime.Now;

public ICollection<PointEntity> Points { get; private set; }
public ICollection<PostEntity> Posts { get; private set; }
public ICollection<LikeEntity> Likes { get; private set; }
public ICollection<TagSelectionEntity> TagSelections { get; set; }

public void Add()
{
    Roles = Roles.User;
    Status = Status.Active;
}

public void ChangeEmail(string address)
{
    Email = new Email(address);
}

public void ChangeUsername(string username)
```

```
        {
            Username = username;
        }

        public void ChangePhone(string phone)
        {
            PhoneNumber = phone ?? string.Empty;
        }
        public void ChangeAbout(string about)
        {
            About = about ?? string.Empty;
        }

        public void ChangeGender(Gender gender)
        {
            Gender = gender;
        }
        public void ChangeSignIn(SignIn signIn)
        {
            SignIn = signIn;
        }

        public void Inactivate()
        {
            Status = Status.Inactive;
        }

        public void AddPoint(PointEntity newPoint)
        {
            Points.Add(newPoint);
        }

        public void AddPost(PostEntity newPost)
        {
            Posts.Add(newPost);
        }
    }
}
using Domain.Post;
using DotNetCoreArchitecture.Domain;
using System;
using System.Collections.Generic;
using System.Text;

namespace Domain.Vote
{
    public class BaseVote
    {
        public long UserId { get; set; }
        public long PostId { get; set; }
        public UserEntity User { get; set; }
        public PostEntity Post { get; set; }
        public DateTime CreationDate { get; set; } = DateTime.Now;
    }
}
using System;
```

```
using System.Collections.Generic;
using System.Text;

namespace Model
{
   public enum Gender
   {
      Male,
      Female
   }
}
using Model.Models.Post;
using System;
using System.Collections.Generic;
using System.Text;

namespace Model.Models.Map
{
   public class MarkerModel
   {
      public long? Id { get; set; }
      public double Altitude { get; set; }
      public double Latitude { get; set; }
      public double Longitude { get; set; }

      public IEnumerable<PostModel> Posts { get; set; }
   }
}
using DotNetCore.AspNetCore;
using DotNetCore.Extensions;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Model;
using Microsoft.AspNetCore.Mvc;

namespace DotNetCoreArchitecture.Web
{
   public abstract class BaseController : ControllerBase
   {
      protected UserModel UserModel => new UserModel
      {
         Id = User.Id(),
         Roles = User.RolesFlag<Roles>()
      };

      public static IActionResult Result(IResult result)
      {
         return ApiResult.Create(result);
      }

      public static IActionResult Result(object data)
      {
         return ApiResult.Create(data);
      }
   }
}
using System;
```

```
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Application.Geolocation;
using DotNetCoreArchitecture.Web;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;

namespace Web.Controllers
{
   [ApiController]
   public class GeolocationController : BaseController
   {
      private readonly IGeolocationApplicationService _geolocationApplicationService;

      public GeolocationController(IGeolocationApplicationService geolocationApplicationService)
      {
         _geolocationApplicationService = geolocationApplicationService;
      }

      // GET api/<controller>/5
      [HttpGet("Countries")]
      [AllowAnonymous]
      public async Task<IActionResult> Countries()
      {
         return Result(await _geolocationApplicationService.GetCountriesAsync());
      }
   }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Application.Post;
using DotNetCore.AspNetCore;
using DotNetCoreArchitecture.Web;
using Microsoft.AspNetCore.Mvc;
using Model.Models.Post;

namespace Web.Controllers
{
   [ApiController]
   [RouteController]
   public class PostController : BaseController
   {
      private readonly IPostApplicationService _postApplicationService;

      public PostController(IPostApplicationService userApplicationService)
      {
         _postApplicationService = userApplicationService;
      }

      [HttpGet("My")]
      public async Task<IActionResult> Get([FromQuery]bool? orderByDateDesc, [FromQuery]bool?
orderByLikesDesc, [FromQuery]string keyword)
```

```
{
    return Result(await _postApplicationService.GetUserPosts(UserModel.Id, new PostsRequest
    {
        Keyword = keyword,
        OrderByDateDesc = orderByDateDesc,
        OrderByLikesDesc = orderByLikesDesc
    }));
}

[HttpGet("{id}")]
public async Task<IActionResult> Get(int id)
{
    return Result(await _postApplicationService.GetPostAsync(id));
}

[HttpGet("{latitude}/{longitude}/{radius}")]
public async Task<IActionResult> Get(double latitude, double longitude, double radius)
{
    return Result(await _postApplicationService.GetMarkerWithPostsNearAsync(
        new Domain.ValueObjects.Coordinate
        {
            Latitude = latitude,
            Longitude = longitude
        },
        radius,
        UserModel.Id));
}

[HttpPost]
public async Task<IActionResult> Post([FromBody]PostModel post)
{
    post.UserId = UserModel.Id;
    await _postApplicationService.CreatePostAsync(post);
    return Result(await _postApplicationService.GetMarkerWithPostsNearAsync(
        new Domain.ValueObjects.Coordinate
        {
            Latitude = post.Marker.Latitude,
            Longitude = post.Marker.Longitude
        },
        0.0001,
        UserModel.Id));
}

//// PUT api/<controller>/5
//[HttpPut("{id}")]
//public void Put(int id, [FromBody]string value)
//{
//}

[HttpDelete("{id}")]
public async Task Delete(int id)
{
    await _postApplicationService.DeletePostAsync(id);
}

[HttpPost("toggleLike")]
```

```
        public async Task<int> ToggleLike([FromBody] VoteModel vote)
        {
            return await _postApplicationService.ToggleLikePostAync(vote.PostId, UserModel.Id);
        }
    }
}
using System;
using System.Collections.Generic;
using System.Linq;
using System.Threading.Tasks;
using Database.Post;
using DotNetCore.AspNetCore;
using DotNetCoreArchitecture.Web;
using Microsoft.AspNetCore.Http;
using Microsoft.AspNetCore.Mvc;
using Model.Models.Post;

namespace Web.Controllers
{
    [ApiController]
    [RouteController]
    public class StatisticController : BaseController
    {
        private readonly ITagSelectionRepository tagSelectionRepository;

        public StatisticController(ITagSelectionRepository tagSelectionRepository)
        {
            this.tagSelectionRepository = tagSelectionRepository;
        }

        [HttpPost("userTypeSeletions")]
        public async Task<IActionResult> AddTagStatistic([FromBody] TagModel tag)
        {
            await tagSelectionRepository.CreateTagSelectionForUser(tag.Tag, UserModel.Id);

            return Ok(await tagSelectionRepository.GetUserTagSelection(UserModel.Id));
        }

        [HttpGet("userPreferredTypes")]
        public async Task<IActionResult> GetUserTypes()
        {
            return Ok(await tagSelectionRepository.GetUserTagSelection(UserModel.Id));
        }
    }
}
using DotNetCore.AspNetCore;
using DotNetCore.Objects;
using DotNetCoreArchitecture.Application;
using DotNetCoreArchitecture.Model;
using Microsoft.AspNetCore.Authorization;
using Microsoft.AspNetCore.Mvc;
using Model.Models.User;
using System.Threading.Tasks;

namespace DotNetCoreArchitecture.Web
{
```

```
[ApiController]
[RouteController]
public class UsersController : BaseController
{
   private readonly IUserApplicationService _userApplicationService;

   public UsersController(IUserApplicationService userApplicationService)
   {
      _userApplicationService = userApplicationService;
   }

   [AllowAnonymous]
   [HttpPost("Register")]
   public async Task<IActionResult> AddAsync(AddUserModel addUserModel)
   {
      var signIn = addUserModel.SignIn;
      await _userApplicationService.AddAsync(addUserModel);
      return Result(await _userApplicationService.SignInAsync(signIn));
   }

   [HttpGet("Profile")]
   public async Task<IActionResult> GetProfile()
   {
      return Result(await _userApplicationService.GetAsync(UserModel.Id));
   }

   [HttpPost("SetAvatar")]
   public async Task<IActionResult> AddAvatarToUserAsync()
   {
      var avatar = ControllerContext.HttpContext.Request.Files()[0];

      await _userApplicationService.SetAvatarAsync(UserModel.Id, avatar);

      return Result(await _userApplicationService.GetAsync(UserModel.Id));
   }

   [AuthorizeEnum(Roles.Admin)]
   [HttpDelete("{id}")]
   public async Task<IActionResult> DeleteAsync(long id)
   {
      return Result(await _userApplicationService.DeleteAsync(id));
   }

   [HttpGet("Grid")]
   public async Task<IActionResult> GridAsync([FromQuery]PagedListParameters parameters)
   {
      return Result(await _userApplicationService.ListAsync(parameters));
   }

   [HttpPatch("{id}/Inactivate")]
   public async Task InactivateAsync(long id)
   {
      await _userApplicationService.InactivateAsync(id);
   }

   [HttpGet]
```

```
        public async Task<IActionResult> ListAsync()
        {
            return Result(await _userApplicationService.ListAsync());
        }

        [HttpGet("{id}")]
        public async Task<IActionResult> SelectAsync(long id)
        {
            return Result(await _userApplicationService.SelectAsync(id));
        }

        [AllowAnonymous]
        [HttpPost("SignIn")]
        public async Task<IActionResult> SignInAsync(SignInModel signInModel)
        {
            return Result(await _userApplicationService.SignInAsync(signInModel));
        }

        [HttpPost("SignOut")]
        public async Task SignOutAsync()
        {
            await _userApplicationService.SignOutAsync(new SignOutModel(UserModel.Id));
        }

        [HttpPut("{id}")]
        public async Task<IActionResult> UpdateAsync(UpdateUserModel updateUserModel)
        {
            await _userApplicationService.UpdateAsync(updateUserModel);
            return Ok();
        }

        [HttpPost("ChangePassword")]
        public async Task<IActionResult> UpdatePasswordAsync(UserChangePassword
changePasswordModel)
        {
            var result = await _userApplicationService.UpdatePasswordAsync(changePasswordModel);

            if (result.Succeeded)
            {
                return Ok();
            }

            return Result(result);
        }
    }
}
using Application.Post;
using DotNetCore.AspNetCore;
using DotNetCore.IoC;
using DotNetCoreArchitecture.Application;
using DotNetCoreArchitecture.Database;
using DotNetCoreArchitecture.Infra;
using Microsoft.AspNetCore.Builder;
using Microsoft.EntityFrameworkCore;
using Microsoft.Extensions.Configuration;
using Microsoft.Extensions.DependencyInjection;
```

```
//using Microsoft.OpenApi.Models;
using System;

namespace DotNetCoreArchitecture.Web
{
   public static class Extensions
   {
      public static void AddClassesMatchingInterfaces(this IServiceCollection services)
      {
         services.AddClassesMatchingInterfaces(typeof(IUserApplicationService).Assembly);
         services.AddClassesMatchingInterfaces(typeof(IPostApplicationService).Assembly);
         services.AddClassesMatchingInterfaces(typeof(IUnitOfWork).Assembly);
         services.AddClassesMatchingInterfaces(typeof(ISignInService).Assembly);
      }

      public static void AddContext(this IServiceCollection services)
      {
         var configuration = services.BuildServiceProvider().GetRequiredService<IConfiguration>();

         var connectionString = configuration.GetConnectionString(nameof(Context));

         services.AddDbContextMigrate<Context>(options => options.UseSqlServer(connectionString));
      }

      public static void AddSecurity(this IServiceCollection services)
      {
         services.AddHash(10000, 128);
         services.AddJsonWebToken("6bzaw-
vTSOZpViwUiKmYG5oLmFcp5l273swV6LNWLlEZSXRDtbf9b7_bDFswKVeS8hE92DdhiZp1KOAhcc
kk4Rq88_1sKlXedUY06_BngPMnTKWTY3MVmGpn96Lxqu2mlE6qIzSu3-
LG9v1yJKct0loTHspDCTd0xx2Aw85xEL0", TimeSpan.FromHours(12));
         services.AddAuthenticationJwtBearer();
      }

      public static void AddSpa(this IServiceCollection services)
      {
         services.AddSpaStaticFiles("Frontend/dist");
      }

      public static void UseSpa(this IApplicationBuilder application)
      {
         application.UseSpaAngularServer("Frontend", "start");
      }
   }
}
using DotNetCore.AspNetCore;
using DotNetCore.IoC;
using Microsoft.AspNetCore.Builder;
using Microsoft.Extensions.DependencyInjection;

namespace DotNetCoreArchitecture.Web
{
   public class Startup
   {
      public void Configure(IApplicationBuilder application)
      {
```

```
            application.UseException();
            application.UseRouting();
            application.UseCorsAllowAny();
            application.UseHttps();
            application.UseAuthentication();
            application.UseAuthorization();
            application.UseResponseCompression();
            application.UseResponseCaching();
            application.UseStaticFiles();
            application.UseEndpoints(endpoints => endpoints.MapControllers());
            //application.UseSpa();
        }

        public void ConfigureServices(IServiceCollection services)
        {
            services.AddLog();
            services.AddCors();
            services.AddSecurity();
            services.AddResponseCompression();
            services.AddResponseCaching();
            services.AddControllers();
            services.AddMvcJson();
            services.AddSpa();
            services.AddFileService();
            services.AddContext();
            services.AddClassesMatchingInterfaces();
        }
    }
}
```

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ГЕОІНФОРМАЦІЙНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ НАДАННЯ РЕКОМЕНДАЦІЙ З ПОШУКУ МІСЦЬ ДОЗВІЛЛЯ

Керівництво користувача

КПІ.ІП-6108.045490.06.34

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Є. А. Недашківський

Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

_____М. О. Денисенко

Київ – 2020 року

Для початку використання системи потрібно автентифікуватися. Це можливо зробити на сторінці Логіну, увівши власний логін та пароль.

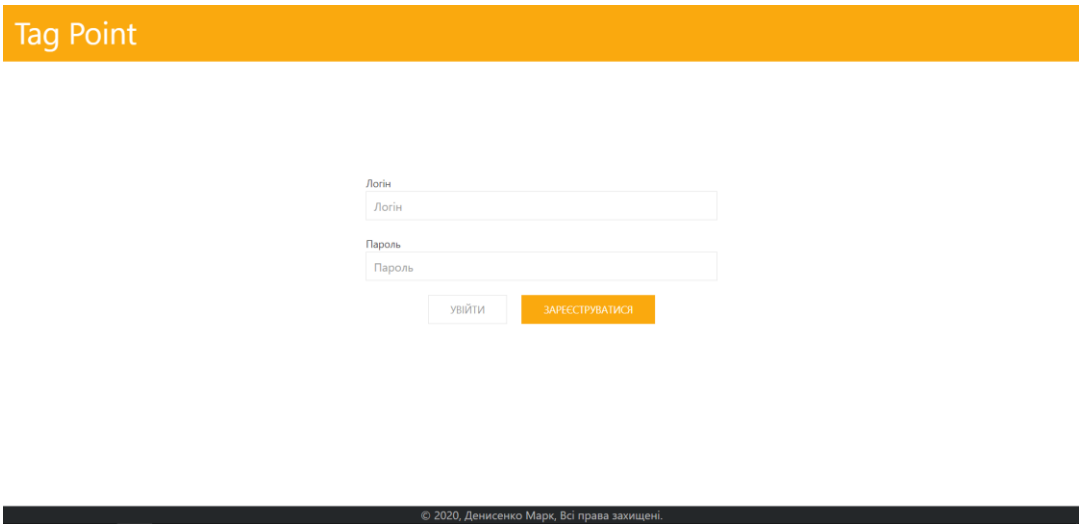


Рисунок 1.1 – Сторінка автентифікації

Нові користувачі можуть перейти на сторінку реєстрації та створити новий профіль. Під час заповнення форми невірно введені поля будуть підсвічені червоним, а коректні зеленим.

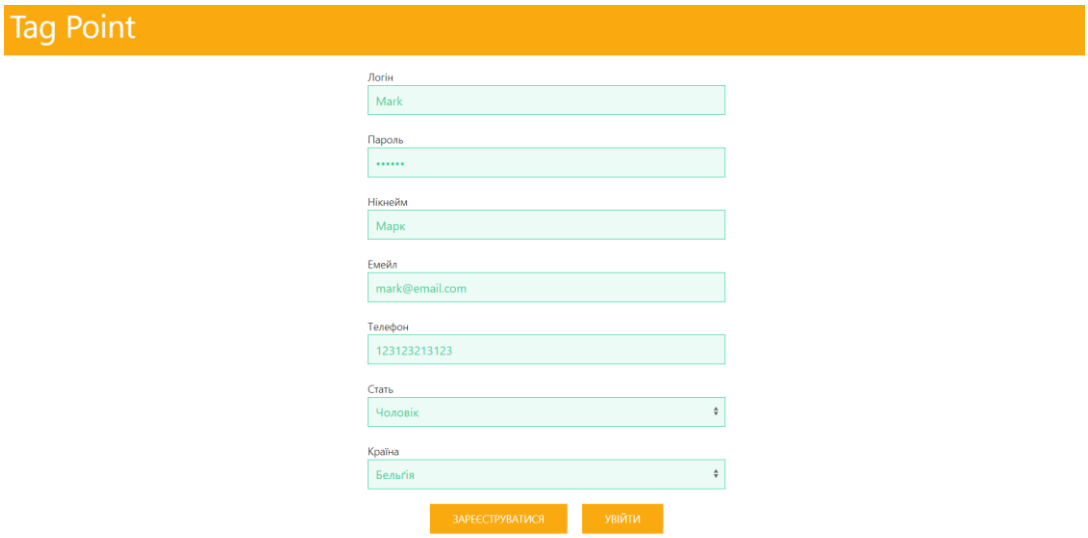


Рисунок 1.2 – Сторінка реєстрації

Після автентифікація або реєстрації користувач буде автоматично перенаправлений на головну сторінку Рисунок 1.3, де наявне навігаційне меню по системі, інтерактивна мапа для пошуку місцевості, список рекомендацій у

обраній точці, можливість отримати персональні рекомендації поблизу та інтерфейс для створення нових рекомендацій.

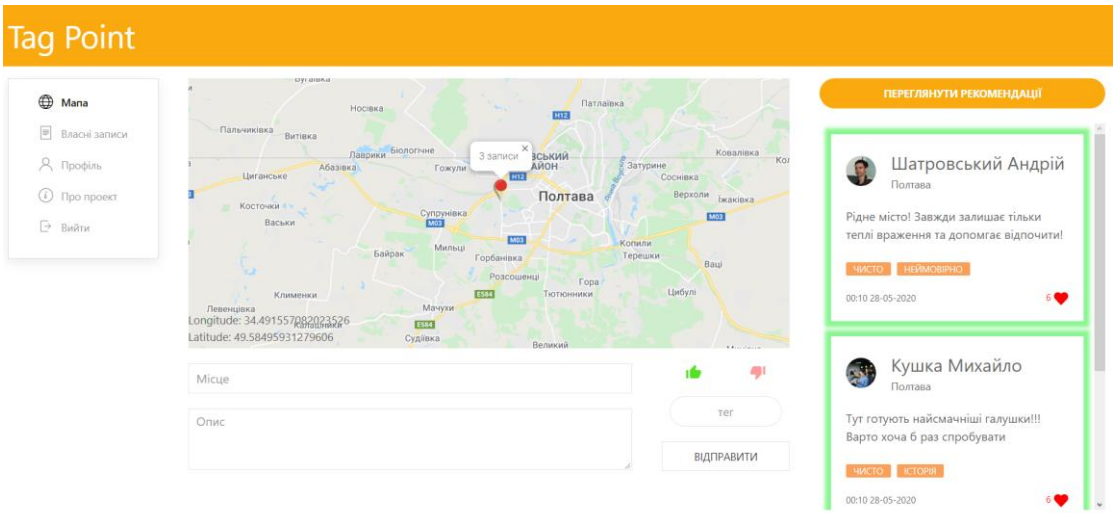


Рисунок 1.3 – Головна сторінка

Натисвнувши «Переглянути рекомендації», користувачеві буде виведено модальне вікно та відображено 3 місця, що найбільше підходять до вподобань.

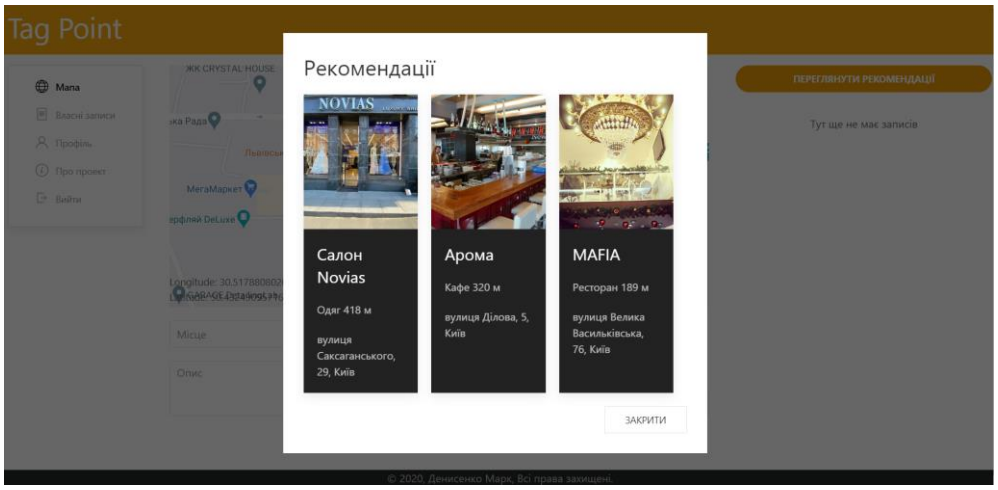


Рисунок 1.4 – Вікно рекомендацій

Для створення власної рекомендації користувачеві необхідно обрати адресу цього місця ввіши або обравши із запропонованих поруч з місцем на

мапі як на Рисунку 1.5.

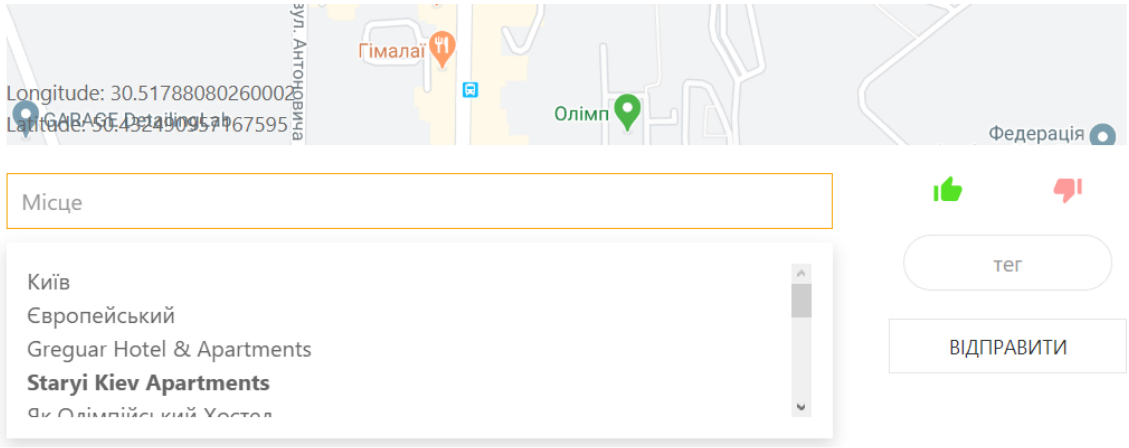


Рисунок 1.5 – Вибір адреси поблизу

За бажанням рекомендацію можна позначити тегом за змістом. Для цього необхідно ввести потрібний тег та натиснути клавішу «Пробіл». Доданий тег буде показаний під повідовленням рекомендації.

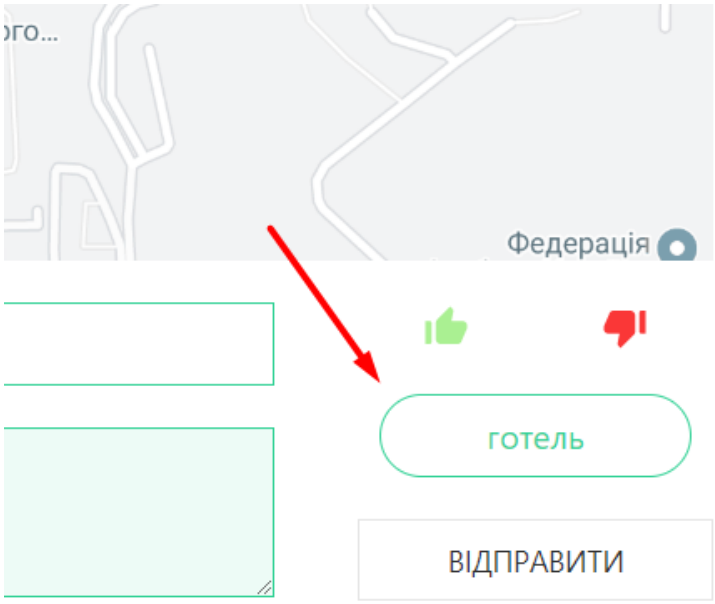


Рисунок 1.6 – Поле для введення тегу

Якщо рекомендація носить негативний характер, то можна позначити її як незадовільну Рисунок 1.7. Така рекомендація буде відображатися в червоній рамці.

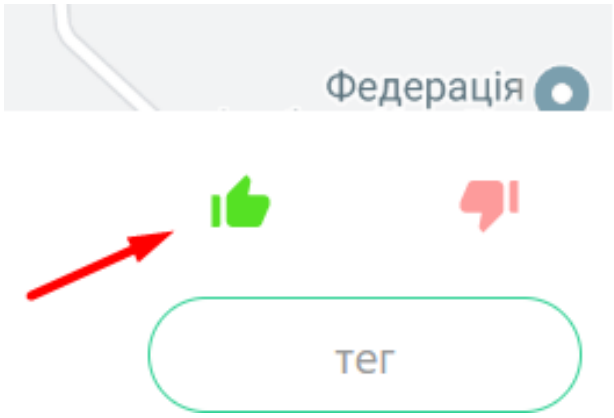


Рисунок 1.7 – Вибір опції рекомендовано або не рекомендовано

Останнім кроком при створенні рекомендації є текстовий опис повідомлення, що буде відображатися. Пусте поле не є допустимим.

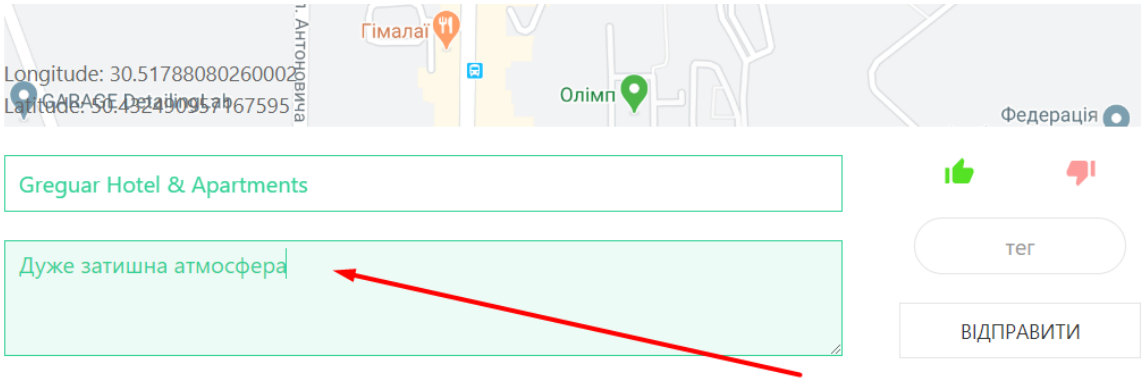


Рисунок 1.8 – Поле введення тексту рекомендації

Коли уся необхідна інформація заповнена, можна створити її у системі натиском на кнопку «Відправити». Після цього на мапі буде створено мітку, що може бути обрана для перегляду. У списку справа від мапи буде відображено нову рекомендацію як показано на Рисунку 1.9.

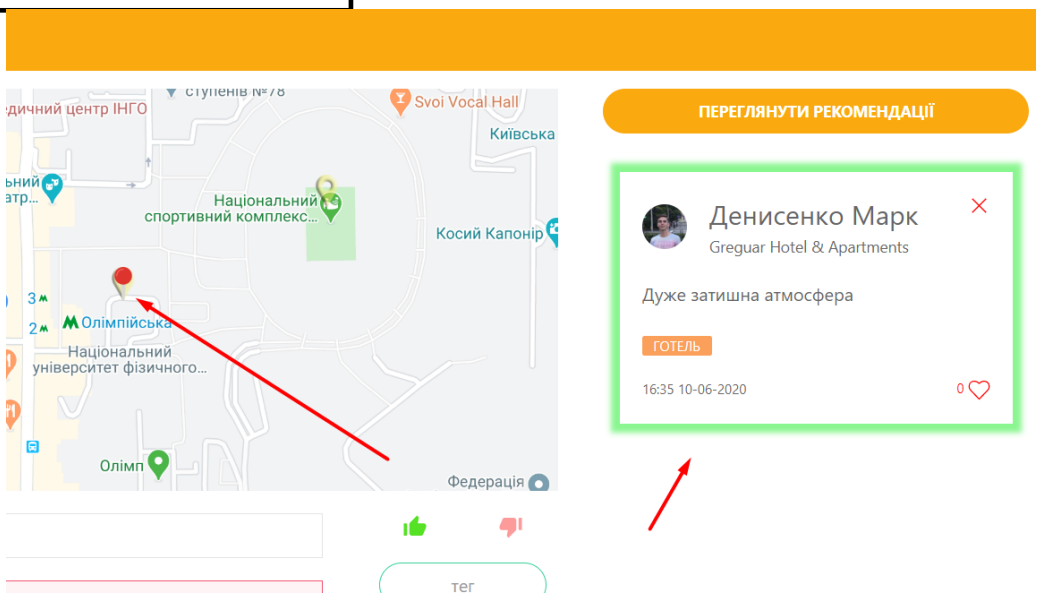


Рисунок 1.9 – Відображення на мапі рекомендації та у списку

Для зручності перегляду та відстежування активності власних рекомендацій можна відкрити сторінку «Власні записи». На ній відображаються усі записи користувача, їх перегляди, вподобання та да створення. За необхідності можна видалити запис. Якщо записів велика кількість, то вони можуть бути відфільтровані за ключовим словом для локації, тексту, перегляду або вподобаннях. Також можна відсортувати записи усіма наявними колонками таблиці.

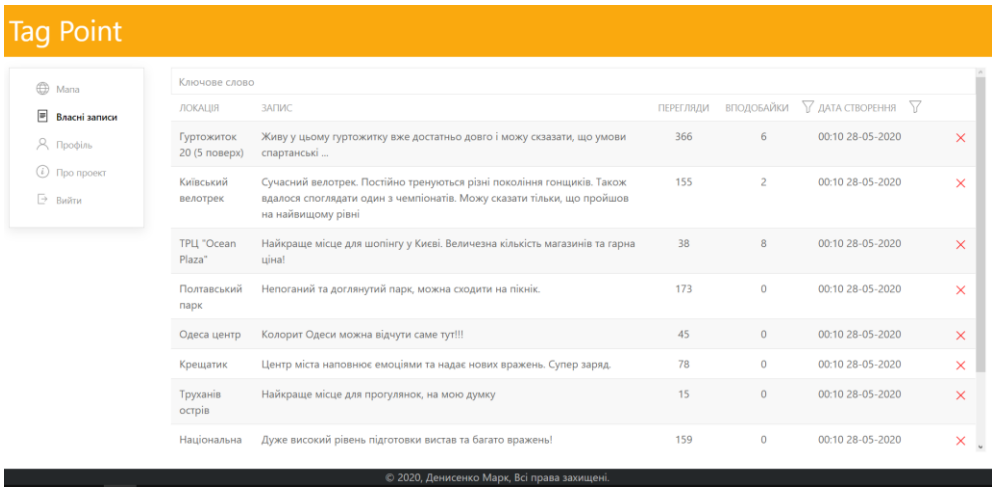


Рисунок 1.10 – Сторінка власних записів

На сторінці профілю користувача показано усю інформацію профілю. Дані можна змінити та зберегти. Також можна завантажити фотографію профілю, натиснувши на місце для фото і обравши власний файл.

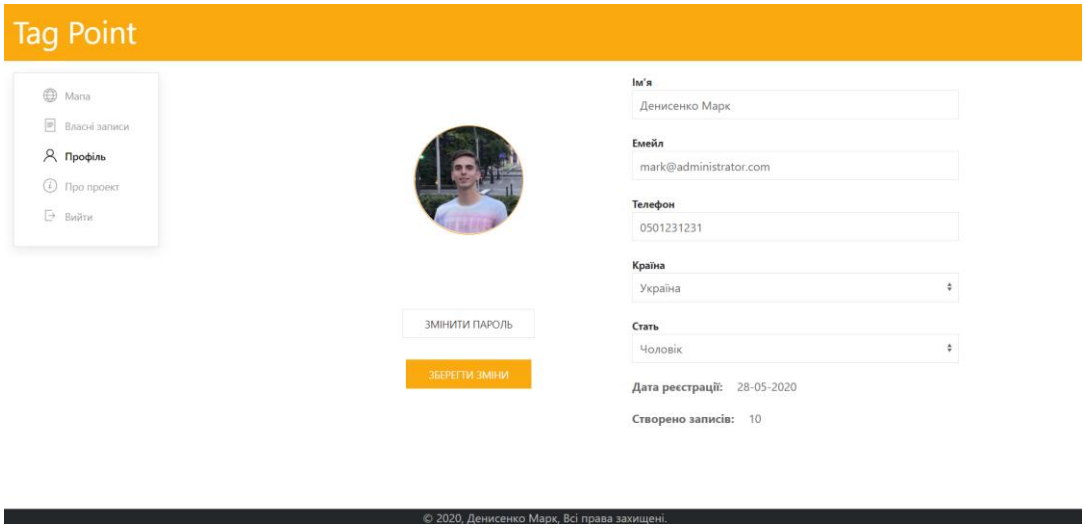


Рисунок 1.11 – Сторінка профілю користувача

Факультет інформатики та обчислювальної техніки

Кафедра автоматизованих систем обробки інформації і управління

“ЗАТВЕРДЖЕНО”

В.о. завідувача кафедри

_____ Олександр ПАВЛОВ

“___” _____ 2020 р.

ГЕОІНФОРМАЦІЙНЕ ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ НАДАННЯ РЕКОМЕНДАЦІЙ З ПОШУКУ МІСЦЬ ДОЗВІЛЛЯ

Графічний матеріал

КПІ.ІП-6108.045490.07.99

“ПОГОДЖЕНО”

Керівник проєкту:

_____ Є.А. Недашківський

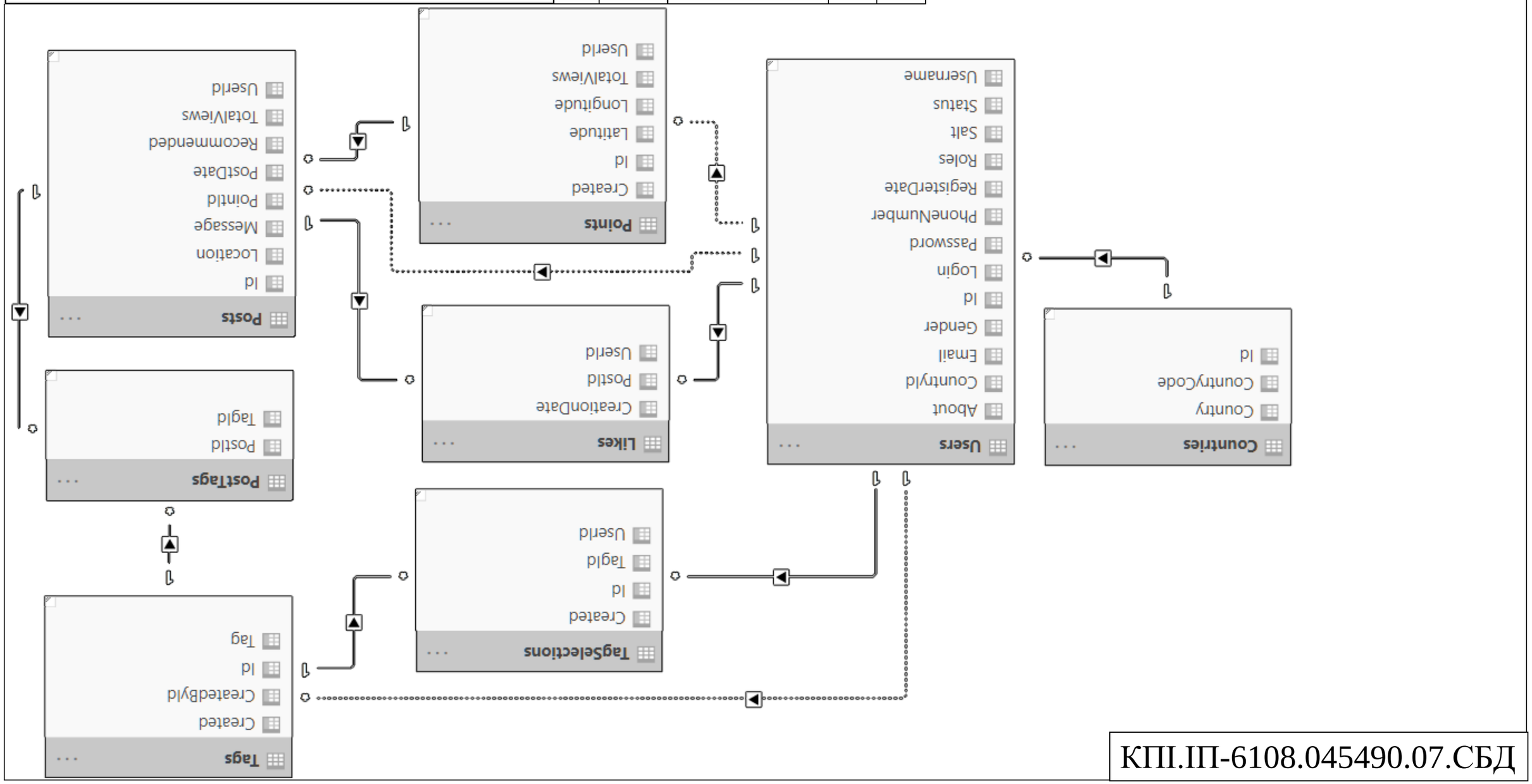
Нормоконтроль:

_____ К.І. Ліщук

Виконавець:

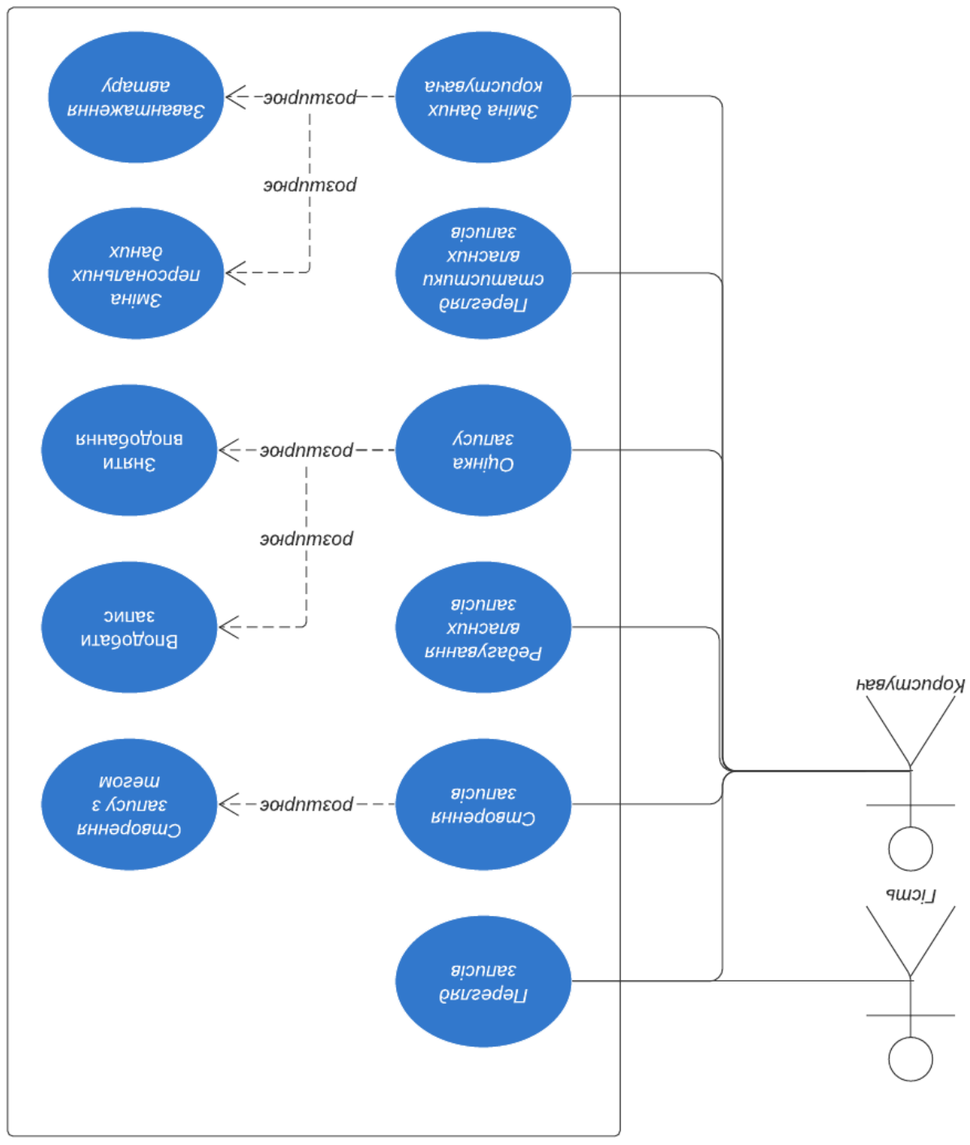
_____М.О. Денисенко

Київ – 2020 року



КПІ.ІП-6108.045490.07.СБД

								КПІ.ІП-6108.045490.07.СБД	
3м.	Арк.	№ докум.	Підпис	Дата					
Розробив		Денисенко М. О.						Схема бази даних	Літ. Маса Масш.
Перевірив		Недашківський Є.А.							
Т. Контр.									Аркуш 1 Аркушів 1
Н. Контр.		Ілцук К.І.						Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61
Затверд.									



КПІ.ІП-6108.045490.07.СС

							КПІ.ІП-6108.045490.07.СС		
Зм.	Арк.	№ докум.	Підпис	Дата					
Розробив		Денисенко М. О.			Схема структура варіантів використань		Літ.	Маса	Масш.
Перевірив		Недашківський Є.А.							
Т. Контр.							Аркуш 1		Аркушів 1
Н. Контр.		Лищук К.І.			Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля		КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61		
Затверд.									

Tag Point

Mana
Власні записи
Профіль
Про проект
Вийти

Ключове слово | ЛОКАЦІЯ | ЗАПИС | ПЕРЕГЛЯДИ | ВПОДОБАЙКИ | ДАТА СТВОРЕННЯ

Гуртожиток 20 (5 поверх)	Жижу у цьому гуртожитку вже достатньо довго і можу сказзати, що умови спартанські ...	366	6	00:10 28-05-2020	×
Київський велотрек	Сучасний велотрек. Постійно тренуються різні покоління тощиків. Також вдалося спогляда0и один з чемпіонатів. Можу сказати тільки, що проїзов на найвищому рівні	155	2	00:10 28-05-2020	×
ТРЦ "Ocean Plaza"	Найкраще місце для шопінгу у Києві. Величезна кількість магазинів та гарна ціна!	38	8	00:10 28-05-2020	×
Поліський парк	Неогляний та доглянутий парк, можна сходити на пікнік.	173	0	00:10 28-05-2020	×
Одеса центр	Колорит Одеси можна відчути саме тут!!!	45	0	00:10 28-05-2020	×
Крещатик	Центр міста наповнене емоціями та надає нових вражень. Супер зарад.	78	0	00:10 28-05-2020	×
Тухляві острів	Найкраще місце для прогулянок, на мою думку	15	0	00:10 28-05-2020	×
Національна	Дуже високий рівень підготовки вистав та багато вражень!	159	0	00:10 28-05-2020	×

Tag Point

Mana
Власні записи
Профіль
Про проект
Вийти

Київ

2 записи

Micце
Опис
тег
ВІДПРАВИТИ

Tag Point

Mana
Власні записи
Профіль
Про проект
Вийти

Ім'я
Денисенко Марк
Емейл
mark@administator.com
Телефон
0501231231
Країна
Україна
Стать
Чоловік
ЗБЕРЕГТИ ЗМІНИ
Дата реєстрації: 28-05-2020
Створено записів: 10

ЗМІНИТИ ПАРОЛЬ

Логін
Пароль
УВІЙТИ
ЗАРЕЄСТРУВАТИСЯ

Tag Point

Mana
Власні записи
Профіль
Про проект
Вийти

Денисенко Марк
Київський велотрек
Сучасний велотрек. Постійно тренуються різні покоління тощиків. Також вдалося спогляда0и один з чемпіонатів. Можу сказати тільки, що проїзов на найвищому рівні
00:10 28-05-2020
2
ГЛОРІ
Шатровський Андрій
Київський велотрек
Найулюбленіше місце для прогулянки. Обожнюю(о
ПЕРЕГЛЯНУТИ РЕКОМЕНДАЦІЇ

Зм.	Арк.	№ докум.	Підпис	Дата	КПІ.ІП-6108.045490.07.КЕ		
Розробив		Денисенко М. О.			Крестенна вигляду екранних форм	Літ.	Маса	Масш.
Перевірив		Недашківський Є.А.						
Т. Контр.						Аркуш 1		Аркушів 1
Н. Контр.		Лищук К.І.						
Затверд.					Геоінформаційне програмне забезпечення для надання рекомендацій з пошуку місць дозвілля	КПІ ім. Ігоря Сікорського Кафедра АСОІУ Група ІП-61		